Efficient Stream Processing of Scientific Data

Thomas Lindemann [1], Jonas Kauke [2], Jens Teubner [3]

DBIS Group, Department of Computer Science
TU Dortmund University, Dortmund, Germany
[1] thomas.lindemann@cs.tu-dortmund.de
[2] jonas.kauke@cs.tu-dortmund.de
[3] jens.teubner@cs.tu-dortmund.de

***Abstract*—Modern particle physics produces volumes of experimental data that challenge any data processing system. To illustrate, the trigger system of the LHCb experiment at CERN must sustain a data rate of 4 TB/s, yet maintain real-time characteristics.**

In this work, we report on *ELPACO*, a distributed event processing platform for scientific data. Its key characteristics are excellent *scalability* and high *resource efficiency*. *ELPACO* inherits its favorable scalability from Apache Storm, which we used as a basis for our platform. For resource efficiency, we tailored *ELPACO* to *Eriador*, a parallel, ARM-based hardware substrate with excellent energy/performance characteristics.

With experiments on realistic data, we confirm a linear scalability (throughput vs. core count) and a 2.5× improvement in energy efficiency compared to existing solutions.

I. Introduction

Particle physics has become a massively data-intensive discipline. Huge particle accelerators—such as the *Large Hadron Collider (LHC)* [1] at CERN—produce vast amounts of experimental data—4 TB/s in the case of the *LHCb experiment* [2] at CERN—which often must be processed in real time. Analyzing these data volumes has become the key limitation of the domain: any improvement in analysis performance translates into better insights on the physics side.

In this work, we report on *ELPACO* (*"Efficient Low-Power Particle Combiner"*), a data processing platform that we built on the basis of Apache Storm. To meet the throughput demands of scientific applications now and in the future, we designed *ELPACO* for maximum *scalability* (with the amount of processing resources). Experiments confirm that *ELPACO* scales linearly also for large core counts.

With (linear) scalability in place, *resource efficiency* becomes a primary concern for large-scale data processing systems (in order to keep cost low; better throughput can always be achieved by adding more cores). With *ELPACO*, therefore, we aim for *(i)* low-cost hardware and *(ii)* high energy efficiency—both aspects that match the characteristics of ARM processor architectures. We tailored *ELPACO* to run on large-scale, low-cost, and low-power ARM installations. On this basis, *ELPACO* achieves a 2.5-fold advantage in power consumption over conventional solutions.

Contributions. The contributions of this paper, therefore, are

(i) we demonstrate how a typical use case from the domain of particle physics can be realized on top of a platform like Apache Storm;
(ii) we report on adaptations, including a tailor-made scheduler, that we applied to Apache Storm to leverage the potential of low-power processors; and
(iii) we experimentally confirm that *ELPACO* meets the set expectations toward high scalability and energy efficiency.

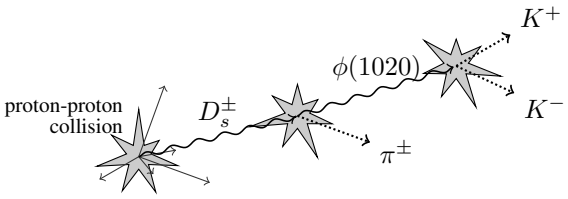

Fig. 1. $D_s^\pm \to \phi(1020)\pi^\pm \to K^+K^-\pi^\pm$ decay channel. A $D_s^\pm$-meson decays into a pion ($\pi^\pm$) and two kaons (K^+ and K^-), which will be seen by the LHCb detector system.

Outline. This paper is structured as follows. Section II sketches the characteristics of typical analysis tasks in the area of particle physics. Section III provides the necessary background on Apache Storm, before we describe the inner workings of *ELPACO* in Section IV. We evaluate *ELPACO* in Section V, before we wrap up in Section VII.

II. The LHCb Experiment at CERN

The *Large Hadron Collider (LHC)* at CERN is the world's largest particle accelerator, located near Geneva (Switzerland). About 100 m under the earth surface, protons are accelerated to near-light speed and then made to collide with one another. As a consequence of the collision, new, unstable particles may form up, but quickly decay into smaller *decay products*.

Figure 1 illustrates this for the decay channel $D_s^\pm \to \phi(1020)\pi^\pm \to K^+K^-\pi^\pm$. A $D_s^\pm$-meson decays into a $\phi(1020)$ and a pion ($\pi^\pm$); the former further decays into two kaons of opposite charge (K^+ and K^-). In practice, the $D_s^\pm$ and $\phi(1020)$ will travel a few centimeters before they decay.

The decay products (here the $\pi^\pm$, K^+, and K^-) can be detected through a series of detectors, which are placed several meters away from the primary collision vertex, as illustrated in Figure 2. [2]

Many possibilities (decay channels) exist according to which the colliding protons might form new particles and

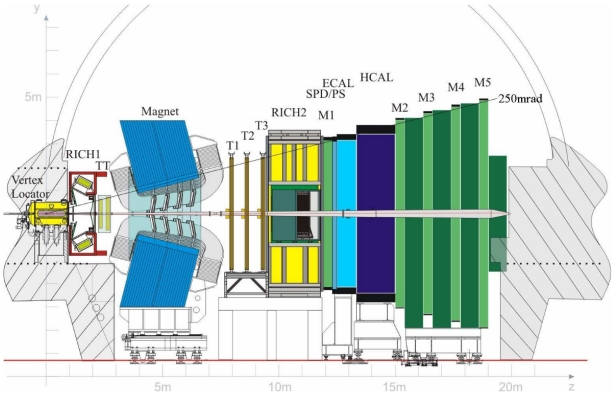

Fig. 2. LHCb detectors. The proton beam is located horizontally in the center of the picture. Protons collide near the "vertex locator" on the left; decay products pass a magnetic field before they are detected in several layers of detectors.

decay afterward. Only few of them, however, are of interest to the physicists (such as the above $D_s^{\pm} \to \cdots \to K^+K^-\pi^{\pm}$ channel). A key part of the analysis, therefore, is to test whether the particles observed by the detectors match a decay channel of interest and filter out others. To this end, recorded energies and particle momentums are added up for each step in the decay channel and according to the rules of physics (preservation of energy and momentum); and the computed mass of the $D_s^{\pm}$-meson is compared to its expected mass ($1968.47 \pm 0.33\,\mathrm{MeV}/c^2$).

This part of the analysis, therefore, acts as a *filter* to the input data stream. But a highly selective one: only 10^{-12} to 10^{-15} of all collisions are "interesting" to the physicists in this sense.

A. Data Characteristics

Given the low probability of observing an "interesting" collision, physicists produce a vast number of collision experiments in the hope of finding a few interesting ones.

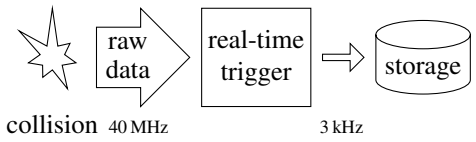

Fig. 3. LHCb trigger system.

The LHCb experiment operates at a frequency of 40 MHz; that is, 40 million collisions are performed every second (year-round). For every collision, detectors record about 100 kB worth of data, resulting in a raw data stream of about 4 TB/s. [3]

This data rate clearly is too large to permanently store all measured collisions. Instead, a *trigger system* (pre-) analyzes the data stream to keep only those data set that (might) match an interesting decay channel.

To illustrate the *ELPACO* platform, in this paper we focus on the $D_s^{\pm} \to \cdots \to K^+K^-\pi^{\pm}$ decay channel mentioned

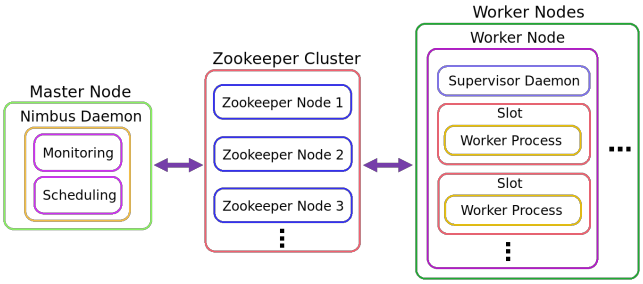

Fig. 4. Storm Architecture

before. The necessary stream filter, slightly simplified, receives a tuple stream of the following schema:

$$\begin{aligned}\mathsf{sch}(\mathit{Track}) = (&\mathit{runNumber}, \mathit{eventNumber}, \mathit{momentumX},\\ &\mathit{momentumY}, \mathit{momentumZ}, \mathit{energy},\\ &\mathit{charge}, \mathit{probNNpi}, \mathit{probNNk})\ ,\end{aligned}$$

where $\mathit{runNumber}$ and $\mathit{eventNumber}$ identify the collision associated with the particle track; $\mathit{momentum}$, energy, and charge denote the particles momentum, energy, and charge; and $\mathit{probNNpi}$/$\mathit{probNNk}$ indicate whether the track is assumed to belong to a pion/kaon (respectively).

In our experiments (Section V) we use a data stream subset that contains 172 million tracks, which are associated to 3.7 million collision events (*i.e.*, this data set, close to 50 particle tracks were recorded for each collision).

B. Analysis Task

To match recorded track information to the decay channels of interest, the analysis framework must first combine candidate tracks that belong to the same collision experiment ($\mathit{runNumber}$, $\mathit{eventNumber}$). From a database perspective, this resembles a join or cogroup operation. Afterward, formulas for energy and momentum preservation must be applied to the combination—an additional join condition in database terms.

III. Apache Storm

Apache Storm [4] is an open-source framework for distributed real-time stream processing. Its characteristics—in particular its potential for scale-out—make it a promising basis for the realization of our *ELPACO* platform. Here we give a short background on the inner workings of Apache Storm, before we discuss how we take advantage of Apache Storm in *ELPACO* (Section IV).

A. Storm Architecture

Figure 4 shows the architecture of a Storm cluster, which consists of master, worker and zookeeper nodes. The master node starts a daemon called *nimbus*. The nimbus receives the topology and schedules the needed executors and tasks to the supervisor slots. In addition the nimbus monitors the supervisors and restarts them if needed.

A worker node starts a daemon called supervisor. The supervisor receives the instructions of the nimbus and starts

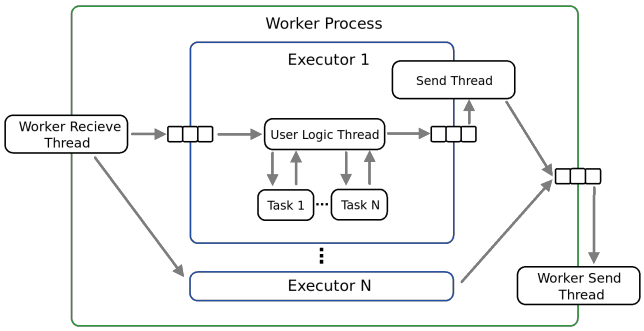

Fig. 5. Dataflow inside a Worker Process.

or stops processes on the node. A supervisor has several slots, where each slot spawns a worker process, which is a JVM (*java virtual machine*). Parts of a Storm topology are executed inside the worker processes. It contains executors and tasks. An executor contains several tasks and got queues for communication with the tasks and the worker process. A task relates to an instance of a spout or bolt, which means all the logic of a topology is processed here. The data flow inside a worker process is illustrated in Figure 5.

Storm topologies are real-time applications, where the data flow of streams is described as a directed graph, with vertices representing computations of the topology and edges representing the transfer and partitioning of stream tuples. The computations of a topology are done in so-called *spouts* and *bolts*. A spout creates a new stream based on an external services like a message broker or an API. Streams are subscribed by bolts, which receive the containing tuples, transform them and generate new streams based on their transformations.

The topology defines the partitioning of a stream for a subscribing bolt with stream groupings. While *shuffle grouping* distributes the tuples of a stream randomly, so that every instance of the bolt receives an equal number of data, *fields grouping* groups the tuples by a field of the tuple. This allows processing of tuples with identical attributes on the same worker node. There are several more stream groupings for different requirements and in addition Storm lets you implement your own logic.

Each zookeeper node runs an instance of Apache Zookeeper [5]. Zookeeper coordinates the communication between the nimbus and the supervisors, which is a key feature to realize failure handling in Storm.

IV. *ELPACO*

To run an analysis for LHCb stream filtering, in *ELPACO* we map the analysis task to a Storm topology. Here we illustrate the concept based on the aforementioned reconstruction of a $D_s^{\pm}$-meson.

Even for simple analysis tasks (such as the decay channel described in this paper), in *ELPACO* we try to break down the data flow in several fine-grained steps/bolts. This allows the underlying Storm framework to better parallelize and scale our workload. The topology graph for our scenario is shown in Figure 6.

In the actual experiment, data may be arriving straight from the data acquisition frameworks. Here we abstract away from these components and prepare all input data in the ORC file format [6], which allows for fast and parallel reading without making the data source (spout) the bottleneck of the entire system.

The *ORCReadingSpout* reads all the input files in parallel and creates a new tuple for every track. Together with the track's unique ID, which describes the assigned event, the spouts emit the tuples to the *GroupEvents* bolt using fields grouping. This bolt collects the incoming tuples and emits a list of tuples of one event to the next bolt, if every tuple was received successfully. This can be checked with the information of the tuple, where the event size is stored additionally to the attributes of the track.

Matching against the $D_s^{\pm} \to \phi(1020)\pi^{\pm} \to K^+K^-\pi^{\pm}$ is implemented in two steps (bolts)—again, to improve parallelization and load balancing: a *CombineKaons* bolt creates candidates for the $\phi(1020)$ particle and forwards them to the *AddPions* bolt. *AddPions* performs the actual matching and yields candidates of the $D_s^{\pm}$-meson (if any are found).

A. Dealing with Low-Power Hardware

For maximum resource efficiency, we designed *ELPACO* to run also on low-power, low-resource hardware (precise hardware information follows in Section V-A).

It turns out, however, that stock configurations of Apache Storm can not properly handle hardware with strong resource limitations—we experienced frequent crashes of individual nodes where often the entire setup could not recover from.

A key challenge is the proper deployment of tasks to ensure an even distribution of load. Standard configurations lack cost information from the application side to find suitable deployments. To this end, we extended Apache Storm by a tailor-made *scheduler*. Our scheduler is well-prepared to deal with constrained hardware and will make sure tasks are assigned evenly to the available processing nodes.

Our scheduler classifies all spouts/bolts into three *slot types*. A *weight* associated with each type defines the number of instances of that type which can be assigned to an execution slot:

(i) The *type 1* category contains all spouts. They perform a lot of disk/SSD I/O; at most one of them should be scheduled on a node.

(ii) Shuffling/grouping operations are categorized as *type 2*. They are relatively light-weight and dominated by communication with peers. Our scheduler lets up to six of them run within a slot.

(iii) Compute tasks (*CombineKaons* and *AddPions* here) perform the actual computation (as seen from the user's perspective). On our system, we found a maximum of two such tasks on a slot to result in good load balancing.

Table I lists the resulting assignment for our sample analysis task.

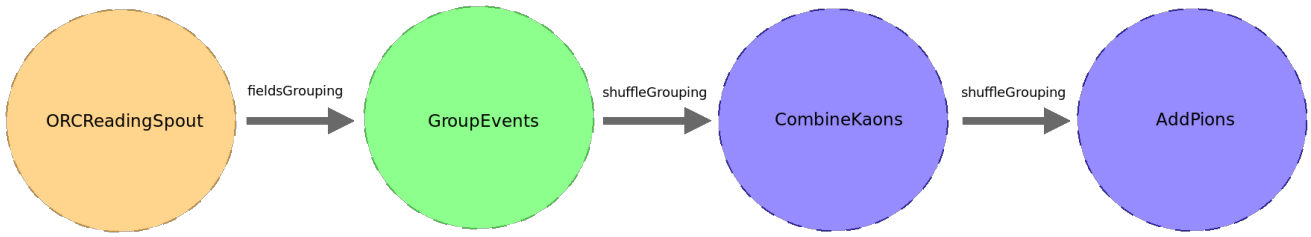

Fig. 6. Topology graph for the execution of our example analysis with *ELPACO*.

TABLE I
INSTANCES PER TYPE OF SLOT.

Bolt	Type 1	Type 2	Type 3
ORCReadingSpout	1	0	0
GroupEvents	0	6	0
CombineKaons	0	0	2
AddPions	0	0	2

TABLE II
HARDWARE CHARACTERISTICS OF OUR TEST SYSTEMS.

Ressource	Intel Xeon	ARM-Cluster
CPU Amount	2	40
CPU Freq.	2.40 GHz	1.50 GHz
Cores	24	160
Threads	48	160
RAM	256 GB	80 GB

V. EVALUATION

To evaluate *ELPACO*, we ran experiments on two very different hardware platforms. Use of a commodity Intel server machine allows to relate our work to existing solutions. A key benefit of *ELPACO* is its ability to run on low-power hardware. With experiments based on an ARM-based cluster, we demonstrate the resulting advantages in terms of resource efficiency.

A. Hardware

Our first system, our "high-performance system", features a dual socket motherboard with two *Intel Xeon CPU E5-2695 v2* processors and a high amount of resources.

The second system is a low-power ARM-Cluster of 40 Odroid-C2 nodes. Each node hosts a *Cortex-A53* 64-bit CPU with—compared to the Intel counterpart—strong resource limitations. The 40 ARM nodes are connected to a gateway, that will run Zookeeper and the Storm nimbus, so the nodes form the heart of the cluster with 40 worker nodes. The hardware components are listed in Table II. On both systems, Storm was running in version 1.1.1 and Zookeeper in version 3.4.10. A network storage connected with 3×1 Gbit/s was used as data source for the spouts.

Table II summarizes the most important hardware characteristics of both our platforms.

As mentioned above, our scheduler will always schedule tasks in groups of one, six, or two, depending on their slot type. In Table III, we listed the resulting number of instances that our scheduler created on the two hardware platforms.

TABLE III
PARALLELIZATION OF THE TOPOLOGY.

Instance	Intel Xeon	ARM-Cluster
ORCReadingSpout	12	12
GroupEvents	36	120
CombineKaons	48	48
AddPions	60	48

B. Measurement Equipment

To compare the efficiency of the test systems, we have chosen to record the energy consumption for processed tuples and the energy-time-product For getting both quality scales, the execution time and the power consumption has to be metered.

In order to meter our reference system, we used a power meter device plugged between the workstation and the mains which is accessible via a network connection. This method has got the disadvantage that the efficiency factor of the power supply is also metered, but it is the least invasive method to meter the overall power consumption.

For metering the boards in our low-power cluster, we used a cluster internal solution which is driven by hall effect current sensors and A/D-Converters with an I2C interface that can be accessed by the Odroid-C2 boards.

C. Experiments

As indicated above, we evaluated *ELPACO* on a data set with 172 million tuples. Here, we report execution times in two components: during a *startup phase*, *ELPACO* will initialize all nodes and workers before it enters the *processing phase*, which reflects the steady-state situation that an actual execution at CERN would observe.

Figures 7 and 8 illustrates an execution of *ELPACO* on our ARM and Intel systems, respectively. The graphs show the progress of time toward the right and CPU utilization along the y axis.

For better readability, the two execution phases are marked in blue and red, respectively. As can be seen, utilization may vary during initialization, but reaches a steady and high value during the actual processing phase.

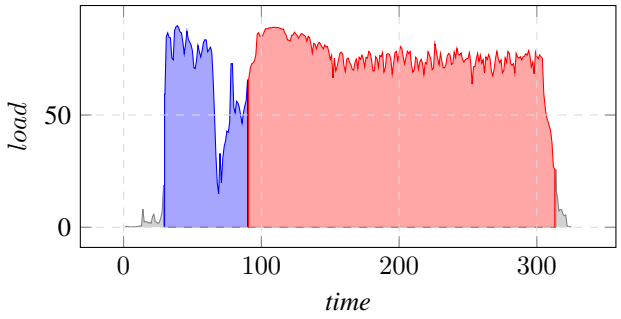

Fig. 7. Execution on ARM-Cluster.

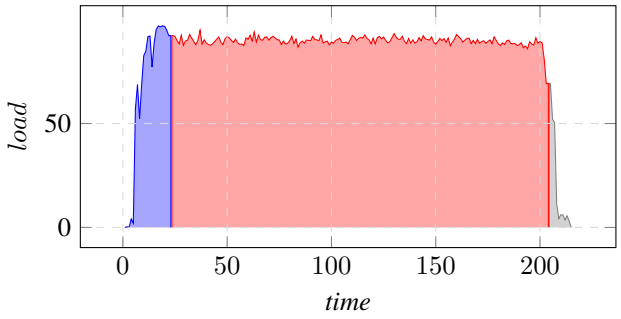

Fig. 8. Execution on Intel Xeon.

To compare *ELPACO* on the two platforms directly, we base our numbers on *(i)* the *total execution time* ("runtime"), which includes the startup and processing phases; *(ii)* the *processing time* only.

Figure 9 shows energy efficiency (represented as *tuples per Joule*) and execution performance (represented as *tuples per second*) for our two platforms.

Most importantly, note that *ELPACO* fares significantly better with respect to energy efficiency. Arguably, the *energy-delay product* (energy consumption multiplied by execution time) better describes the energy efficiency of a system (since it considers both performance and energy consumption). In Table IV, we listed both metrics. Observe how *ELPACO* can profit from the ARM hardware and achieve a 2.5 times better energy-delay product.

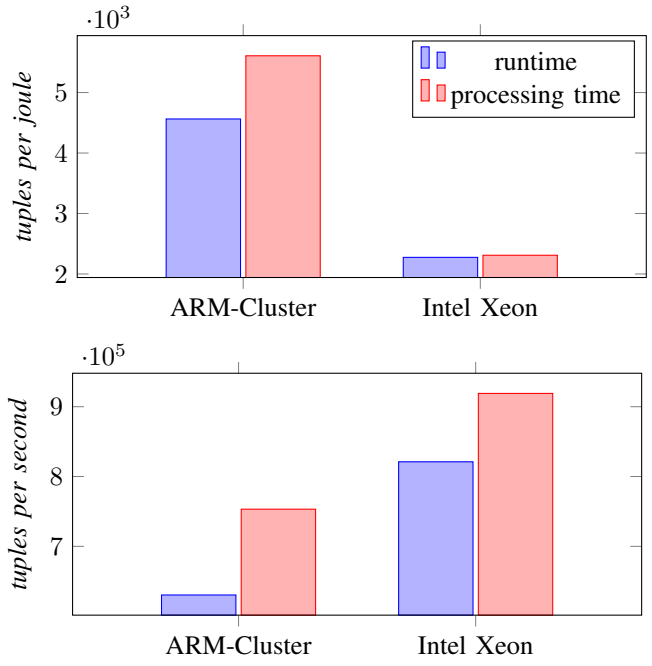

Fig. 9. Comparison of systems.

TABLE IV
COMPARISON OF SYSTEMS

Indicator	Run Time		Processing Time	
	Intel	Cluster	Intel	Cluster
tuples per second	821,099	630,345	919,037	753,236
average power [W]	360.95	128.80	397.79	134.27
EDP	$15.9 * 10^6$	$11.0 * 10^6$	$14.0 * 10^6$	$6.2 * 10^6$
tuples per Joule	2275	4563	2310	5608

D. Scalability

A key design goal of *ELPACO* is to achieve (linear) performance scalability with the amount of resources used for processing. To evaluate whether we achieved this goal, we scaled *ELPACO* to run on a varying number of nodes within our ARM cluster.

Figure 10 shows the resulting throughput rates. As can be seen in the figure, *ELPACO* indeed meets our goal of linear scalability. This means that, by adding more compute nodes, *ELPACO* could easily be configured to meet the throughput demands of the experiments at CERN.

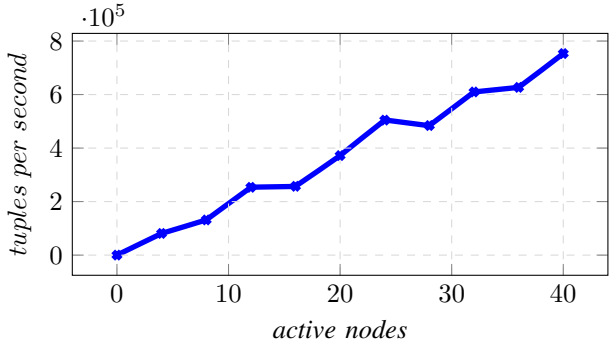

Fig. 10. Run time of *ELPACO* over node count.

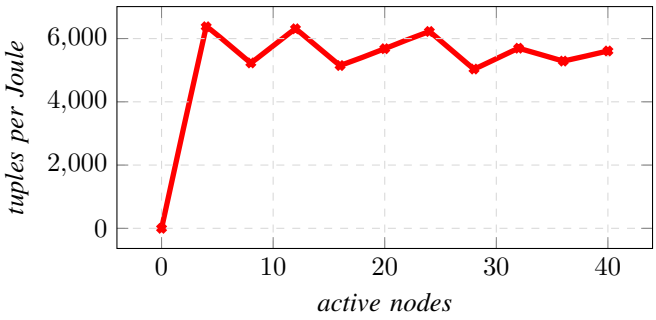

Fig. 11. Efficiency of *ELPACO* over Node count

ELPACO, thereby, does not lose its favorable resource efficiency characteristics. Figure 11 shows how the *tuples per Joule* metric changes as the system is scaled to larger node counts. As can be seen, already for small configurations the system reaches its full research efficiency (and does not degrade afterward).

VI. Related Work

Stream processing emerged as a new processing paradigm in the database community in the early millennium (*Aurora* [7] is a well-known example), and distributed execution soon became attractive (*e.g.*, in the form of the *Borealis* [8] prototype). These systems still lacked the ability to scale out to very large configurations, a property that Google's *MapReduce* [9] paradigm could provide, albeit only in the form of batch-oriented processing.

Inspired by use cases from social networks, adaptations of the MapReduce idea to stream processing engines were demonstrated, including Twitter's *Storm* [10]. (later also *Heron* [11]); Facebook's *Puma*, *Swift*, and *Stylus* [12]; *Drizzle* [13]; or *Apache Flink*/*Stratosphere* [14]. Systems like *BigDAWG* [15] marry stream- and batch-oriented processing.

In *ELPACO*, we carefully respect the characteristics of our ARM-based, low-energy platform. Resource awareness was also the goal of the *AdaStorm* system of Weng *et al.* [16], which adaptively reacts to fluctuating workloads. Pohl *et al.* [17] studied modern hardware characteristics in the context of data stream processing.

Scientific applications adapt the new processing models only slowly. The *Kira* system [18] uses Apache Spark in an astronomy image processing, achieving linear scale-out similar to *ELPACO*. Choi *et al.* [19] uses a stream-oriented processing model to analyze Fusion Energy experiments. Within CERN, one initiative currently aims to use Apache Spark for the accelerator's logging service.

VII. Conclusion

Our experiments have shown that our test cluster hardware optimized for energy efficiency has with 66% less energy consuption a lower energy footprint, but only 15% less processed tuples per second. Overall, we got a factor about 2.6x better energy efficiency in tuples per joule than a state-of-the-art Xeon dual socket reference server system. Thus, architectures that are optimized for energy efficient processing are a good alternative to conventional big data systems. As the system is scalable to the needed performance, you can still benefit from the reduced power consumption for different workload sizes. While distributing complex applications over multi core architectures is a challenge, Apache Storm is a very useful framework to parallelize our processing.

Acknowledgment

The authors would like to thank the collaboration partners from the Experimental Physics Department E5A of the TU Dortmund University in the SFB876-C5 collaboration, which is provided by the Research Project SFB876.

References

[1] L. Evans and P. Bryant, "Lhc machine," *Journal of Instrumentation*, vol. 3, no. 08, p. S08001, 2008.

[2] The LHCb Collaboration, "The lhcb detector at the lhc," *Journal of Instrumentation*, vol. 3, no. 08, p. S08005, 2008.

[3] "LHCb Trigger and Online Upgrade Technical Design Report," Tech. Rep. CERN-LHCC-2014-016. LHCB-TDR-016, May 2014. [Online]. Available: https://cds.cern.ch/record/1701361

[4] Apache Software Foundation, "Apache Storm," https://storm.apache.org, last visited 06.11.2017.

[5] ——, "Apache Zookeeper," https://zookeeper.apache.org, last visited 06.11.2017.

[6] ——, "Apache ORC," https://orc.apache.org, last visited at 06.11.2017.

[7] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik, "Aurora: A new model and architecture for data stream management," *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[8] Y. Ahmad, B. Berg, U. Çetintemel, M. Humphrey, J. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, and S. B. Zdonik, "Distributed operation in the Borealis stream processing engine," in *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Baltimore, MD, USA, Jun. 2005, pp. 882–884.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *6th Symp. on Operating System Design and Implementation (OSDI)*, San Francisco, CA, USA, Dec. 2004, pp. 137–150.

[10] Twitter Inc., "Storm@twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 147–156. [Online]. Available: http://doi.acm.org/10.1145/2588555.2595641

[11] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter Heron: Stream processing at scale," in *Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data*, Melbourne, Vic, Australia, May 2015, pp. 239–250.

[12] G. J. Chen, J. L. Wiener, S. Iyer, A. Jaiswal, R. Lei, N. Simha, W. Wang, K. Wilfong, T. Williamson, and S. Yilmaz, "Realtime data processing at Facebook," in *Proc. of the 2016 ACM SIGMOD Int'l Conf. on Management of Data*, San Francisco, CA, USA, Jun. 2016, pp. 1087–1098.

[13] S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. J. Franklin, B. Recht, and I. Stoica, "Drizzle: Fast and adaptable stream processing at scale," in *Proc. of the 26th Symp. on Operating Systems Principles (SOSP)*, Shanghai, China, Oct. 2017, pp. 374–389.

[14] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke, "Nephele/PACTs: A programming model and execution framework for web-scale analytical processing," in *Proc. of the 1st ACM Symposium on Cloud Computing (SoCC)*, Indianapolis, IN, USA, Jun. 2010, pp. 119–130.

[15] J. Meehan, S. Zdonik, S. Tian, Y. Tian, N. Tatbul, A. Dziedzic, and A. J. Elmore, "Integrating real-time and batch processing in a polystore," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, Sep. 2016, pp. 1–7.

[16] Z. Weng, Q. Guo, C. Wang, X. Meng, and B. He, "AdaStorm: Resource efficient Storm with adaptive configuration," in *Proc. of the 33rd IEEE Int'l Conf. on Data Engineering (ICDE)*, San Diego, CA, USA, Apr. 2017, pp. 1363–1364.

[17] C. Pohl, P. Götze, and K.-U. Sattler, "A cost model for data stream processing on modern hardware," in *Proc. of the 8th Int'l Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*, Munich, Germany, Sep. 2017.

[18] Z. Zhang, K. Barbary, F. A. Nothaft, E. R. Sparks, O. Zahn, M. J. Franklin, D. A. Patterson, and S. Perlmutter, "Scientific computing meets big data technology: An astronomy use case," *CoRR*, vol. abs/1507.03325, 2015. [Online]. Available: http://arxiv.org/abs/1507.03325

[19] J. Y. Choi, T. Kurc, J. Logan, M. Wolf, E. Suchyta, J. Kress, D. Pugmire, N. Podhorszki, E.-K. Byun, M. Ainsworth, M. Parashar, and S. Klasky, "Stream processing for near real-time scientific data analysis," in *Proc. of the 2016 New York Scientific Data Summit (NYSDS)*, New York City, NY, USA, Aug. 2016, pp. 1–8.