Live Cell Recognition and State Classification

Justin Baraboo, Bill Capps, Brendan O’Connor, and David Rodgers

***Abstract*—The efficiency of Big Data frameworks is constantly evolving to process data as it is acquired. Many frameworks had previously focused on just batch processing or streaming data, often going offline to update a model. Newer frameworks have shown that a combination can be achieved with a lambda architecture. In the field of microbiology, microscopy video data is moving towards a real-time requirement for faster analysis of data. In our investigation, we explore the application of a lambda architecture using Apache Spark, Apache Storm and Kafka with streaming microscopy video data to achieve a real-time Big Data classification framework of yeast and white blood cells.**

Keywords—microscopy; yeast; white blood cells; big data; lambda architecture; Storm; Spark

I. Introduction

The recent progress in big data analytics has been significant. Our ability to efficiently handle datasets of larger size has increased the field of applications. This is also met with the ability to handle data processing at speeds unthinkable before. We attempted to harness these new advances in the field to solve real time data processing problems. For this experiment we focused exclusively on the domain of cell microscopy data. Our goal was to produce a proof of concept system that could help automate routine microscope lab work. The end result hopefully freeing researchers time to work on more impactful work. For the purposes of this experiment we focused exclusively on cell classification. We did so with the view that once this could be done reliably we could extend the system to achieve more meaningful tasks such as cell counts and tracking the lifecycle and lineage of each cell. What we found in attempting to do so was the tools to achieve this are readily becoming available and practical. The framework is in place to implement the system, however, we have yet to show meaningful results in regards to classification. We suspect this could be achieved by refining the machine learning techniques used. For the scope of this experiment we only had time to try one approach, however, with different approaches and more data the classifications could likely lead to meaningful and useful results.

II. Related Works

A. Cell Microscopy Video Analysis

Cell microscopy can benefit from a real time data solution. Cells have been able to be classified [1,2], tracked in live time [3], and had key actions, such as cell reproduction, identified [4]. Live tracking utilized SIFT points of cells of low-contrast differential interference contrast (DIC) microscopy; these points underwent principal component analysis and used a Laplacian Eigenmap to map cells between frames. These works did not work in a real time system; Jiang’s work, while working in live time, utilizes only Matlab for their architecture, limiting ability to funnel in and work on data as well as being a proprietary software. His work also maps out cell tracking between frames very nicely which would be extremely useful for lineage generation of cells by mapping. Each of these works is really a part of the sum of what would be needed for a real time system for classification, tracking, and action recognition.

B. Lambda Architecture

One of the very first milestones of Big Data Analytics was the creation of Hadoop and the MapReduce framework [5,6]. With the MapReduce framework, the data is split up into partitions and undergoes two phases, Map and Reduce [7]. These two phases are the building blocks to many tasks and help to achieve a fairly efficient and fast performance that is unparalleled near the time of its creation. But it has some serious pitfalls. The growing complexity of Big Data problems of today is too great for the MapReduce with severe overhead and CPU issues. More importantly, MapReduce frameworks cannot sufficiently match the needs of data that needs be to processed and analyzed in real-time. The lambda architecture is a real-time architecture that can be applied to a multitude of frameworks, with its ability to handle both streaming, real-time data and also batch data (similar to MapReduce) to create a singular, adaptable model. In this review, we take a look at a variety of frameworks to illustrate the scalability, adaptability and efficiency of the lambda architecture.

Apache Spark is a framework that is similar to MapReduce while still supporting a lambda architecture. By itself, it is a micro-batch processing framework that can outperform MapReduce’s efficiency. It has the Resilient Distributed Dataset (RDD), that is immutable and helps to allow fast access to data. It is also fault-tolerant and helps to reduce overhead while running MapReduce operations [8]. It works well in a lambda architecture with Storm or Heron. Storm and Heron are both primarily stream processing engines that take data continuously (in the form of tuples) and distribute them to bolts to carry out tasks [9]. Heron is a much more complex, but also more efficient and fault-tolerant than Storm [10]. These streaming engines have the capability of incorporating their input and output with Spark to create simultaneously a stream and batch framework, not unlike the lambda architecture [11]. It provides the capability to create a model that can both learn from stored datasets as well as new data coming in, especially with Kafka. Kafka is a publisher-subscriber system to which data can be streamed and from which data can be consumed [12]. This allows for a user to publish data, and have their Storm-Heron-Spark lambda architecture consume that data in real-time. It can also be the broker between Spark and Storm, and serve as a reliable, scalable, efficient piece of the lambda architecture, without which the key elements of streaming and batch processing would not be able to efficiently combine to tackle the scale of Big Data.

The trend to support both large scale batch and stream processing evolves with other frameworks, altering the architecture to handle a variety of data and environments. For instance, Pipeline61 framework is designed to be compatible with a multitude of execution environments, while S4 uses Processing Elements to handle data in any order or type while achieving high accuracy and speed [13,14]. Both of these frameworks open potential to create complex lambda architectures to not just combine streaming and batch processing, but also a variety of data, offering perhaps a realistic approach to Big Data solutions. Summingbird is another framework that could work well with others to create a scalable lambda architecture [15]. Its most powerful feature is to bring together online and batch MapReduce operations, which is well tailored to a Big Data lambda architecture.

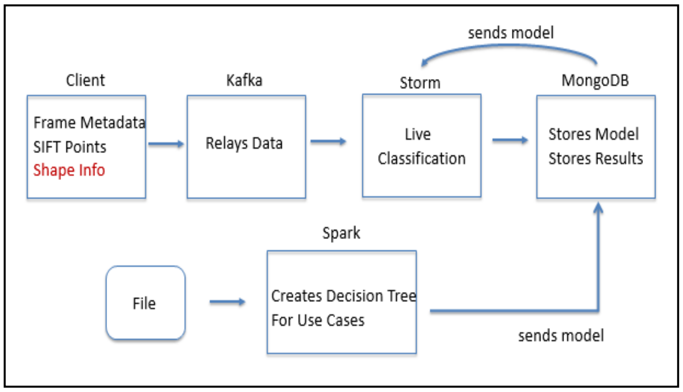
Many may think that the downfall of having any streaming portion of an architecture is that it relies on already trained data or a model that has been built. Even if the model adapts to new data, it implies that the new data may not be as powerful as the data collected from a batch process. While this is not necessarily true, there are two frameworks that combine trained and untrained data to create a mostly unsupervised learning approach to address this very issue. Long Short Term Memory and Recurrent Neural Networks and Deep Learning of Traffic Flows are studies that have one major concept in common: auto-encoders. Auto-encoders play an important role in both reconstruction and prediction of data, and are especially important in neural network flexibility [16,17]. As a result, these networks can learn an adaptable model with primarily streaming data as its basis for its model [18]. While also having a lambda architecture, it can prioritize at once what it has learned and what is actual useful based on previous and incoming data; its usefulness has noted with predicting video sequences and traffic flows.

Other applications take on much larger scale issues that affect us on a daily scale. Cloud computing incorporates a lambda-like architecture in creating a synergistic robot to compute both pre-mapped models and generate new movement patterns based online data [19]. WiseReplica works well with video-on-demand solutions that require adaptability based on user input and patterns to reduce replication and increase video bitrate [20]. Such efficiency is owed in large part to its lambda architecture, being able to adapt to new information while retaining an overall model or goal. Scientists were also able to combine a variety of sources of data as well as a complex mix of Machine Learning algorithms to be able to predict human emergency mobility following natural disasters (such as an earthquake) [21]. While this model does not entirely implement a streaming engine of any sort, the fact that it can process both visual, audio, numerical, etc., data in large quantities shows that it has the adaptability and scalability of a lambda-like architecture, for these different data types cannot generate a similar model with a MapReduce-like framework. Indeed, the power of flexibility, scalability, and efficiency of these Big Data frameworks can only be achieved with a lambda architecture.

III. Implementation

Our lambda architecture consists of Spark, Storm, and Kafka [Fig. 1] which receives data live from the client and uses a MongoDB database as an intermediary and final storage for model and results. Spark is used for our offline learning and model building, where we used Sparks MLLib to create a decision tree. Storm is used for online classification where different bolts provide true and false answers for classifying cells as white blood cells or yeast cells. Kafka and to an extent MongoDB acts as the intermediary communication channel between Spark and Storm and the client.

Fig. 1. Lambda Architecture Design.



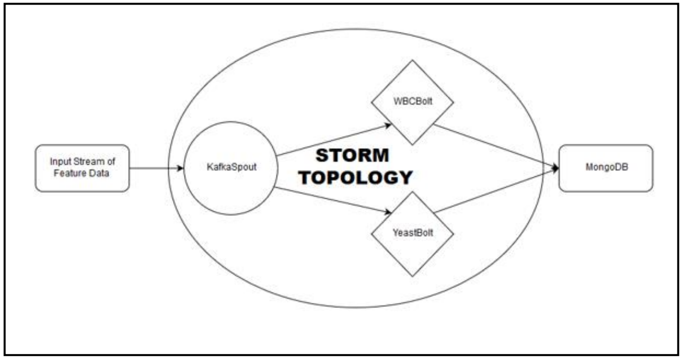
Client sends data to Kafka which stores it to be streamed into Storm for live classification and results stored in mongodb. Spark is used for offline machine learning which sends the computed model to mongodb to be stored and then retrieved to be parsed into bolts for Storm's classification.

Our real time classification workflow consists of the client extracting key frames and sift points from the microscopy image and sending them to Kafka. Kafka queues this data and then acts as a spout for Storm to pull this data from and perform classification. The tuple of the form {frame metadata, SIFT point} are streamed into each bolt in our topology [Fig. 2] and are emit true or false based on the classification to the MongoDB with the frame metadata which allows for validation.

Our model was trained in Spark, where we extracted SIFT points from 20 yeast cell images and 10 white blood cell images, totaling around 500 SIFT points each. The images were converted to grayscale before SIFT extraction and principal component analysis was not applied. We then used Spark's MLLib to create a decision tree to decide on cell classification. The model is then sent to the MongoDB and parsed to create bolts for Storm.

We tested our model on 5 videos, 3 white blood cell and 2 yeast, with over ten thousand SIFT points extracted over each class. The source code for the project and a video demo are included in the references [43,44].

Fig. 2. Generated Storm Topology.



Tuples of frame metadata and SIFT points are streamed in from the Kafka spout and flow into each classification bolt. The results of the classification are sent the mongodb. This forms the basic classification aspect of our design and each bolt could be further extended into multiple state and action recognition bolts once the cell has been classified.

IV. Results and Evaluation

TABLE I. MICROSCOPY VIDEO SUMMARY

Video Metadata	*Class*	
	Yeast	***White Blood Cell***
Length (s)	19	87
Data Points	16810	12300
Number of Frames	1268	2514
Run Time (mm:ss)	3:40	3:45

[a] Summary of Microscopy Videos – Two videos were used for Yeast cells, and three videos were used for White Blood Cells

In our evaluation, we used three videos of white blood cells and two videos of yeast cells. Table 1 shows the summary of this microscopy video data. Our model achieved a 20% error rate with training data.

TABLE II. CONFUSION MATRIX – WHITE BLOOD CELL

Confusion Matrix: White Blood Cell		*Predicted*	
		White	***Not White***
Actual	***White***	655	15638
	Not Yhite	269	12657

b. Confusion Matrix for White Blood Cell Classification

TABLE III. CONFUSION MATRIX – YEAST CELL

Confusion Matrix: Yeast Cell		*Predicted*	
		Yeast	***Not Yeast***
Actual	***Yeast***	110	16536
	Not Yeast	97	12376

c. Confusion Matrix for Yeast Cell Classification

The white blood cell classifier performed about as same chance, with an almost equal accuracy and error rate. It seems overall this classifier misclassifies a lot of actual white blood cells as not white blood cells (recall = 4.02%). When it does predict white blood cells, however, it usually is fairly accurate (precision = 70.9%). Given by the low recall, it seems that the classifier needs to increase its sensitivity to improve performance.

TABLE IV. CLASSIFICATION STATISTICS

Parameters	*Class*	
	Yeast	***White Blood Cell***
Accuracy	45.6	42.9
Error	54.4	57.1
Recall	4.02	0.66
Precision	70.9	53.1

d. Classification Statistics for Yeast and White Blood Cells: All values are percentages

The yeast classifier performed also just a little bit worse than chance (accuracy = 42.9%). The yeast classifier tends to misclassify actual yeast cells are not yeast cells (low recall < 1%), but when it does classify a SIFT point as a yeast cell, it performs a little bit better than chance (precision = 53.1%). Overall, this classifier had a low sensitivity, which would need to improve to improve the overall performance.

Taken together, both classifiers performed similar and tend to be less sensitive than desired. Given this, it would seem that there is more of a systemic error that is involved with the SIFT feature extraction for the model building and/or for testing.

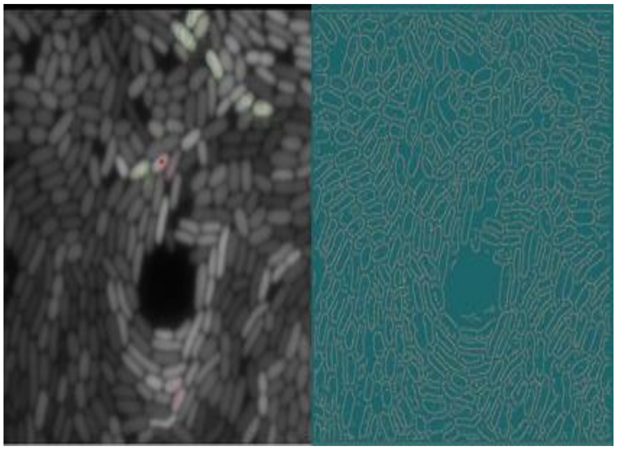
V. CONCLUSION

We were successfully able to implement Spark, Storm, and Kafka as a real time data solution to ingest live cell video data and generate classifications. We were able to build decision tree models within Spark using the MLLib library, use Storm for online classification, and Kafka as the communicator bridge between them. Our parsing of the model from spark is dubious as the confusion matrix of the test data contains many unclassified points between the bolts which could never happen in a single decision tree. Whether this is a parser error or a code error is unknown at this point. It does seem that once points are classified, white blood cells can be differentiated, and yeast cells are classified a little better than random; however, the bulk of the points remain unclassified. The time taken process the points is a little too long; however, if aggregation schemes are utilized, we would see this reduced as our feature size per frame would also reduce. Our work does sets up a good architectural groundwork of processing cell microscopy in real time.

VI. FUTURE WORK

Our project can be extended from and improved upon its initial vision, setting the stage for much larger bodies of work. To better our results, we need to fix the interaction between MongoDB and bolt creation. Furthermore, we need to aggregate our SIFT points using some aggregation scheme; Fisher Vector is a great scheme for this as cells could have a distinct Gaussian Mixture Model in SIFT Space. Also, applying dimension reduction via principal component analysis to our SIFT points for model building would be beneficial. SIFT points are 158 dimensional objects and the majority of the information and variation of the data is probably included in only a few of these. (Eigenvalue analysis when doing PCA could verify this).

Fig. 3. Canny Filter applied to an Image of E. coli.



Shape data could also be utilized for action and state recognition. We are currently in the process of implementing a canny filter to find edges of our cell images (Fig. 3). Since certain cells like yeast and E. coli tend to have well defined shapes, circular and ovular to amorphous, it’s possible to classify them with that as well; furthermore, state in reproduction cycle is also shape based as yeast buds and E. coli pinches off in the middle forming a different shape.

Finally, the future of our work is vast. Finding the lineage of a cell by applying cell tracking to classification would allow researchers to find births and deaths of certain cells with certain characteristics. Also, live classification of cell state would allow a more modular approach to biological labs as determining results wouldn’t have the wait until after the experiment and allow many steps to be performed instead of just one per experiment.

REFERENCES

[1] Chen, Zhou & Wong. “Automated segmentation, classification, and tracking of cancer cell nuclei in time-lapse microscopy.” IEEE Transactions on Biomedical Engineering, vol. 53, no. 4, April 2006. http://ieeexplore.ieee.org/document/1608529/

[2] Goutam & Sailaja. “Classification of acute myelogenous leukemia in blood microscopic images using supervised classifier.” 2015 IEEE International Conference on Engineering and Technology (ICETECH), March 2015. http://ieeexplore.ieee.org/document/7275021/

[3] Jiang, RM et alia. “Life-Cell Tracking Using SIFT Features in DIC Microscopic Videos.” IEEE Transactions on Biomedical Engineerings vol57.no5.September 2010.

[4] Tran, Pham & Zhou. “Cell phase identification using fuzzy Gaussian mixture models.” Proceedings of 2005 International Symposium on Intelligent Signal Processing and Communication Systems, 2005. http://ieeexplore.ieee.org/document/1595447/

[5] "What is big data?" IBM - Bringing Big Data to the Enterprise. URL: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

[6] Woodle, A. “From Elephants to Spiders: The History of Hadoop”. https://www.datanami.com/2015/04/15/from-spiders-to-elephants-the-history-of-hadoop/

[7] Dean J, Ghemawat S. "MapReduce: simplified data processing on large clusters". Communications of the ACM Magazine - 50th anniversary issue: 1958-2008, Vol 51: Issue 1, Jan 2008.

[8] Zaharia, Matei, et al. "Spark: cluster computing with working sets." HotCloud 10 (2010): 10-10.

[9] Toshniwa A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D. "Storm @Twitter". Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp 147-156, 2014.

[10] Kulkarni, Sanjeev, et al. "Twitter heron: Stream processing at scale." Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015.

[11] Bijnens N. "A real-time architecture using Hadoop and Storm". DataCrunchers, Dec 2013. URL: http://lambda-architecture.net/architecture/2013-12-11-a-real-time-architecture-using-hadoop-and-storm-devoxx.

[12] Kreps J, Narkhede N, Rao J. "Kafka: a Distributed Messaging System for Log Processing". NetDB Workshop, 2011.

[13] Wu, Dongyao, et al. "Building Pipelines for Heterogeneous Execution Environments for Big Data Processing." IEEE Software 33.2 (2016): 60-67.

[14] Neumeyer, Leonardo, et al. "S4: Distributed stream computing platform." 2010 IEEE International Conference on Data Mining Workshops. IEEE, 2010.

[15] Boykin, Oscar, et al. "Summingbird: A framework for integrating batch and online mapreduce computations." Proceedings of the VLDB Endowment 7.13 (2014): 1441-1451.

[16] Srivastava N, Mansimov E, Salakhutdinov R. "Unsupervised Learning of Video Representations using LSTMs", Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pp 843-852, 2015.

[17] Lv, Yisheng, et al. "Traffic flow prediction with big data: a deep learning approach." Intelligent Transportation Systems, IEEE Transactions on 16.2 (2015): 865-873

[18] Zhang W, Xu L, Li Z, Lu Q. "A Deep-Intelligence Framework for Online Video Processing". IEEE Computer Society, pp 44-51, Mar 2016.

[19] Bekris K, Shome R, Krontiris A, Dobson A. "Cloud Automation: Precomputing Roadmaps for Flexible Manipulation", IEEE Robotics & Automation Magazine Vol 22: Issue 2, 2015.

[20] Silvestre S, Buffoni D, Pires K, Monnet S, Sens P. "Boosting Streaming Video Delivery with WiseReplica". Transactions on Large-Scale Data-and Knowledge-Centered Systems, 34-58.

[21] Song, Xuan, et al. "A Simulator of Human Emergency Mobility following Disasters: Knowledge Transfer from Big Disaster Data." Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015.

VIDEO REFERENCES

A. *Training Videos for Yeast*

[22] https://www.dropbox.com/sh/gtslv0u1l6hgvlq/AAAa96AsSGDkiPHCzvuil7pBa?dl=0

These videos were gathered from YouTube. Randomly chosen frames were used and cropped to specific Yeast Cells in Microsoft Paint. Using Window’s 10 Photo Processor, all images were converted to 100% grayscale.

B. *Training Videos for Yeast*

[23] https://www.youtube.com/watch?v=PTGYLYru2Sw

[24] https://www.youtube.com/watch?v=5vWDqZu-iHk

[25] https://www.youtube.com/watch?v=YzRpAfYArQg

[26] https://www.youtube.com/watch?v=CbuPYxtM0Y8

[27] https://www.youtube.com/watch?v=cfTomGVcZYc

[28] https://www.youtube.com/watch?v=sFZkDM2Dv7U

[29] https://www.youtube.com/watch?v=9Mg-WqE-TwY

[30] https://www.youtube.com/watch?v=YMKbRPIAFdw

[31] https://www.youtube.com/watch?v=0wtA3p6kqtA

[32] https://www.youtube.com/watch?v=N8bp1FC6myc

These videos were gathered from YouTube. Randomly chosen segments between four and 40 seconds were spliced from the video. Each video was converted to grayscale FFmpeg. This reduced variation in background between videos to reduce variability in SIFT Feature extraction.

C. *Training and Testing Videos for White Blood Cells*

[33] https://www.youtube.com/watch?v=Va1jaBGwoT8

[34] https://www.youtube.com/watch?v=zgJNhhtIAvg

[35] https://www.youtube.com/watch?v=JnlULOjUhSQ

[36] https://www.youtube.com/watch?v=sYCUWqc_x3U

[37] https://www.youtube.com/watch?v=PsrnMeu6sIY

[38] https://www.youtube.com/watch?v=qvGVoxdy-yM

[39] https://www.youtube.com/watch?v=2TKTSHrw5QI&t=2s

[40] https://www.youtube.com/watch?v=KAnFGn-ecNU

[41] https://www.youtube.com/watch?v=rj-MS4Oxdu8

[42] https://www.youtube.com/watch?v=f0UGnI5N8lg

These videos were gathered from YouTube. Randomly chosen segments between four and 40 seconds were spliced from the video. Each video was converted to grayscale FFmpeg. This reduced variation in background between videos to reduce variability in SIFT Feature extraction.

D. GitHub Repository

[43] Source: https://github.com/DRinKC/RT-BigData-Project_Team1

[44] Demo: https://github.com/DRinKC/RT-BigData-Project_Team1/tree/master/Demo

The source code can be viewed in segments to indicate milestones and progression by feature. The demo is divided into two videos and shows the full implementation.