International Journal on Data Science and Technology
2021; 7(2): 32-39
http://www.sciencepublishinggroup.com/j/ijdst
doi: 10.11648/j.ijdst.20210702.12
ISSN: 2472-2200 (Print); ISSN: 2472-2235 (Online)

Kognitor: Big Data Real-Time Reasoning and Probabilistic Programming

Arinze Anikwue*, Boniface Kabaso

Information Technology Department, Cape Peninsula University of Technology, Cape Town, South Africa

Email address:
anikwuea@cput.ac.za (A. Anikwue), kabasob@cput.ac.za (B. Kabaso)

*Corresponding author

To cite this article:
Arinze Anikwue, Boniface Kabaso. Kognitor: Big Data Real-Time Reasoning and Probabilistic Programming. *International Journal on Data Science and Technology*. Vol. 7, No. 2, 2021, pp. 32-39. doi: 10.11648/j.ijdst.20210702.12

Received: June 10, 2021; **Accepted**: July 9, 2021; **Published**: August 2, 2021

Abstract: There is a huge increase in the amount of generated data since the explosion of the Internet. This generated data which is usually collected in different formats and from multiple sources is popularly termed Big Data. Big data contains uncertainty. To handle uncertainty in big data, probabilistic reasoning is used to develop probabilistic models that specify generic knowledge in different topics. These models are used in conjunction with an inference algorithm to enable decision makers especially during uncertain situations. Extensive knowledge in fields such as statistics, machine learning and probability theories are employed in the development of these probabilistic models. Thus, it is usually a difficult undertaking. Probabilistic programming was introduced to simplify and enable development of complex models. Again, decision makers often need to use knowledge from historic data as well as current data to make cogent decisions. Thus, the necessity to unify processing of historic and real-time data with low latency. The Lambda architecture was introduced for this purpose. This paper presents a framework called Kognitor that simplifies the design and development of difficult models using probabilistic programming and Lambda architecture. Evaluation of this framework is also presented in this paper using a case study to highlight the crucial potential of probabilistic programming to achieve simplification of model development and enable real-time reasoning on big data. Thus, demonstrating the effectiveness of the framework. Finally, results of this evaluation are presented in this paper. The Kognitor framework can be used to steer effective and easier implementation of complicated real-life situations as probabilistic models. This will be beneficial in the big data processing domain and for decision makers. Kognitor ensures cost-effectiveness using contemporary big data tools and technology on commodity hardware. Kognitor framework will also be beneficial in academia with respect to the use of probabilistic programming.

Keywords: Big Data Processing, Probabilistic Model, Lambda Architecture, Probabilistic Programming

1. Introduction

Planning, analysis and/or calculation generate facts, that are usually not organized. A collection of these facts can be referred to as data. Huge amount of data is produced and amassed, sometimes as a secondary product from the activities and processes of entities and individuals [1]. This huge data is termed big data. There are three popular qualities of big data: variety, high velocity, and high volume, also known as the 3Vs [2]. The 3Vs have formed the foundational definition of big data.

Data is generated from multiple, unidentical sources with distinct level of uniformity [3]. This disparity in uniformity introduces noisy data that must be efficiently managed using novel techniques such as artificial intelligence and machine learning [4, 5]. To address uncertainty in big data, the machine learning research community introduced probabilistic reasoning. Probabilistic reasoning uses probability theory with deductive logic to enable formal reasoning especially in varying conditions [6–9]. Probabilistic reasoning is also used to interpret complicated situations to make decision making easier [10–12].

The process of decision making is now automated. Automated systems that aid in decision making are generally called probabilistic reasoning systems [13]. According to [14], probabilistic reasoning systems consist of probabilistic models and inferences algorithms. A probabilistic model is an all-inclusive generic information and components of a domain

encoded in probability theories such as Bayesian network [15], hidden Markov models [16, 17] and stochastic grammar [18–21]. Probabilistic models as well as evidence are used by inference algorithms to produce probabilistic score as a response to queries. This procedure is known as probabilistic inference [14].

Designing a probabilistic model is a vigorous task that requires extreme technical expertise in fields such as natural language, mathematics, algorithms [8, 11, 22]. Again, many real-life situations are too expensive to model [14, 23–26]. In response to these issues, probabilistic programming emerged through efforts from the machine learning and programming language communities [24, 27].

According to [14, 24, 26], probabilistic programming is a relatively recent idea. Nevertheless, [28] advocates the capabilities of probabilistic programming in artificial intelligent systems. This paper presents a framework that demonstrates the effectiveness of probabilistic programming in the development of big data processing systems that uses complex probabilistic models. This paper also showcases the constructive integration of off the shelf, open-source contemporary tools and technology for big data on commodity hardware to realize Lambda architecture.

The rest of the paper is organized as follows. Section 2 presents the background knowledge around processing of big data, Lambda architecture, probabilistic reasoning, and probabilistic programming. In Section 3, similar projects/research is presented. Section 4 describes the Kognitor framework. Section 5 demonstrates an implementation of the framework using a case study. An evaluation of the framework is provided in Section 6. This paper ends with a conclusion presented in Section 7.

2. Background

Novel tools and techniques are required for the effective management and analysis of big data. Early technologies developed to analyze big data were mainly geared towards batch processing [29]. The majority of these batch processing tools used the MapReduce framework designed by Google [30]. A popular example of the batch processing big data tool implemented using MapReduce is Hadoop. Hadoop became widely accepted and extensively used in academia and industry [31–35]. Although MapReduce and Hadoop presented advantages in big data processing, they were unsuitable for processing high speed big data that requires low latency [36–40]. Thus, the need for big data stream processing.

Stream processing sometimes referred as real-time processing deals with the velocity attribute of big data. Stream processing handles small pieces of data, thereby enabling low latency [29, 41–43]. Some examples of open-source stream processing systems are Apache Storm [44], Apache Spark [45] and SQLstream [46].

Decision makers require the processing of both static (high volume) and real-time data together for well informed decision making [47, 48]. Thus, stream processing or batch processing tools in isolation may not be the remedy in real-life situations. The need for solutions that support batch and real-time big data processing is apparent [49–51]. Solutions that support this combination are called hybrid computation. Some examples are Kappa architecture [50], the Liquid architecture [52] and Lambda architecture [49].

2.1. Lambda Architecture

Lambda architecture was designed and proposed by [49] as a hybrid solution to big data issues. This architecture is made up of three layers – the batch layer, the serving layer, and the speed layer. Each of the three layers is responsible for a unique problem in big data. The functionalities of these layers are built on each other. The batch layer is the central part of Lambda architecture. Raw or unprocessed data is permanently stored in the batch layer. The unprocessed data in the batch layer is periodically processed using a batch processing framework to yield batch views. To balance the high latency batch processing, the speed layer employs incremental model to achieve real-time processing. Result from the processing in the speed layer is known as real-time views. As soon as processing on the same dataset is done in the batch layer, the corresponding real-time views are consequently discarded. The serving layer uses both the batch views and real-time views to provide low latency response to user queries [53–58].

2.2. Probabilistic Reasoning

The process of decision making is sometimes straightforward, but in other cases, decision making may require complicated procedures which involves evidence from many sources [59]. This is clearly seen in uncertain circumstances where the odds of uncertain events influences decision making [13, 60]. The eventuality of an event is represented using probability. Thus, probabilistic reasoning simplifies decision making using underlying principles or knowledge and probability. Probabilistic reasoning is an integration of what holds true about a circumstance with the laws of probability [14].

Probabilistic reasoning systems are applications that automate the process of probabilistic reasoning. A probabilistic reasoning system is typically made up of an inference algorithm and a probabilistic model [14]. Probabilistic reasoning systems are useful for prediction, deduction, and improvement of general knowledge in a domain.

2.3. Probabilistic Programming

Due to the difficulty and complexity associated with modelling real-life scenarios as probabilistic models, the concept of probabilistic programming was introduced.

Probabilistic programming uses the powerful characteristics of programming language to facilitate the representation of complex probabilistic models [14, 23, 61]. Thus, instead of expressing probabilistic models using Bayesian networks or hidden Markov models, procedures or

functions are used [12]. The automation of inference computation to handle uncertainty is also made easier using probabilistic programming [26, 27, 61, 62]. Church [51], Anglican [63], IBAL [64], BLOG [65], PRISM [66], and Figaro [14, 23] are some examples of probabilistic programming systems.

3. Related Work

It is believed that probabilistic programming would ease the process of designing complex probabilistic models. Research conducted by [67] to examine or measure the adoption of probabilistic programming in big data processing resulted in the identification of a solution called InferSpark [68]. At the time of publication, InferSpark claimed to be the only solution that uses probabilistic programming to provide efficient statistical inference on big data. The authors of InferSpark also recognized the potential of probabilistic programing in the development of complex probabilistic models, while pointing out drawbacks of contemporary probabilistic programming systems.

An evaluation of InferSpark was provided. According to [68], InferSpark outperformed MLib, and Infer.NET. However, InferSpark implemented only one inference algorithm called Variation Messaging Passing (VMP). The framework presented in this paper improves on this using a probabilistic programming system called Figaro. Figaro allows the design of a probabilistic model using either Bayesian network, Markov models, or a combination of both. In addition, the Kognitor framework demonstrates and achieves low-latency computation by implementing Lambda architecture.

4. System Architecture

A major motivation for the Kognitor framework is the need to support real-time decision making in uncertain situations. Kognitor uses of probabilistic programming to enable easier development of real-life complex models, and Lambda architecture to achieve real-time big data processing with low latency. This framework also achieves cost-effective data processing using a combination of contemporary off-the-shelf tools and technologies on commodity hardware. Kognitor is made up of three components namely feeder, server, and storage. The three layers of Lambda architecture are implemented in the three components of Kognitor framework.

4.1. Feeder Component

This component is responsible for data ingestion into Kognitor. The flow of data into Kognitor is managed by the feeder component. Data from multiple sources can be aggregated in the feeder component. The feeder component also cleans up and filters unnecessary and unrelated data before persisting in the storage component.

4.2. Storage Component

The storage component houses data used by Kognitor framework. This component is made up of the master, pseudo-master, batch-view, and realtime-view databases. Each of these databases is responsible for a unique storage need of Kognitor.

The master database is responsible for storing immutable, continuously expanding data. Thus, it should support batch reads and random writes. This forms an implementation of the batch layer as proposed by Lambda architecture. The pseudo-master database holds data as it arrives from the feeder component.

The batch-view and realtime-view databases are used to store result of data processing done on the master and pseudo-master databases, respectively.

In accordance with the Lambda architecture, the master database actualizes an implementation of the batch layer, the batch-view database actualizes part of the serving layer while the pseudo-master and realtime-view databases implement part of the speed layer.

4.3. Server Component

The central component of the Kognitor framework is the server component. All data processing is done by the server component. The server component is sub-divided into two modules: the batch module and the real-time module.

The batch module performs computation on the data stored in the master database. This computation happens at a set time interval. On the other hand, as soon as data is available in the pseudo-master database, the real-time module performs computation on data.

The batch module implements part of the batch layer of Lambda architecture, while the real-time module implements part of the speed layer.

It is important to note that there are two types of computations done in the server component. The first is the learning computation. Kognitor uses an algorithm to learn from the data stored in both the master database and the pseudo-master database. Results from the learning computation done on master database are stored in the batch-view database while results from learning computation on pseudo-master data are stored in the realtime-view database.

The second type of computation is the reasoning computation. Kognitor uses an inference algorithm alongside data from the batch-view and realtime-view databases to perform reasoning computation. Figure 1 shows all the components in Kognitor framework.

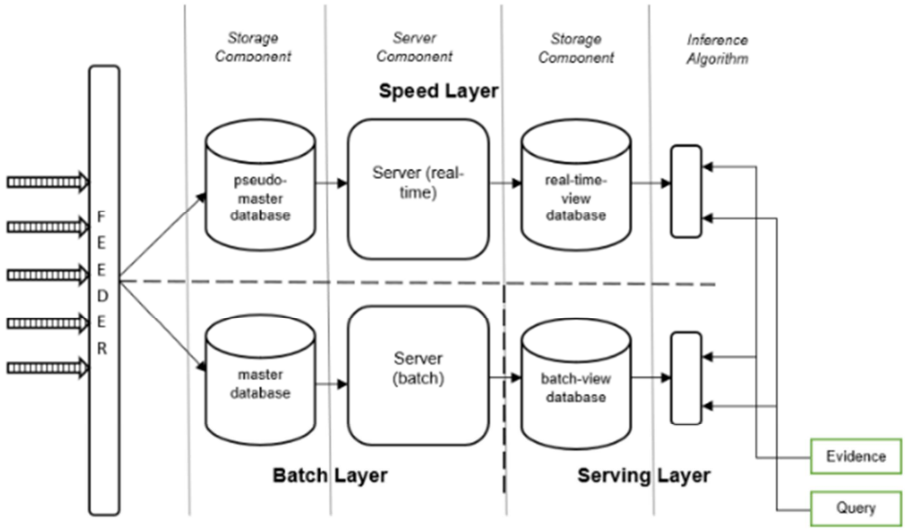

Figure 1. *The Kognitor Framework.*

5. Case Study

To show the effectiveness of Kognitor framework, an application called K4F was developed using Kognitor framework. K4F predicts the outcome of a football match. In this case study, two football teams were selected from the English Premiership League.

5.1. Feeder Implementation

Akka [69, 70] was used to implement the feeder component of Kognitor in K4F. A mock repository was used as a source of data for K4F. An Akka actor was implemented to act as a pipeline between the data repository and K4F. Another Akka actor was implemented to persist the data from the pipeline into the master and pseudo-master databases.

5.2. Storage Implementation

The storage component of Kognitor was implemented using Apache Cassandra [71] in K4F. In the master database, four tables were created to handle the storage need of K4F. The tables were team, rating, form, and fixture. The pseudo-master database also consists of same tables as in the master database. Tables were also created in the batch-view and realtime-view database to hold results of computations by the server component.

5.3. Server Implementation

In K4F, the server component was implemented using Figaro. Figaro represents probabilistic models using elements (variables), relationships between these elements, the functional parameters of the element relationships, and the numerical form of the functional parameters. In this case study, four variables were chosen to represent an indication of a win in a football match. The chosen elements are:

a. *Has Good Rating:* A Boolean variable dependent on a team's rating. The rating can take a value between 0 and 10, 10 being the highest (best) rating.
b. *Has Good Form:* A Boolean variable dependent on a team's performance in their last six (6) games.
c. *Has Home Ground Advantage:* A Boolean element dependent on a team's performance when in their home ground.
d. *Is Winner:* A Boolean element indicating the possibility of a win.

The relationship between the chosen elements is shown in Figure 2.

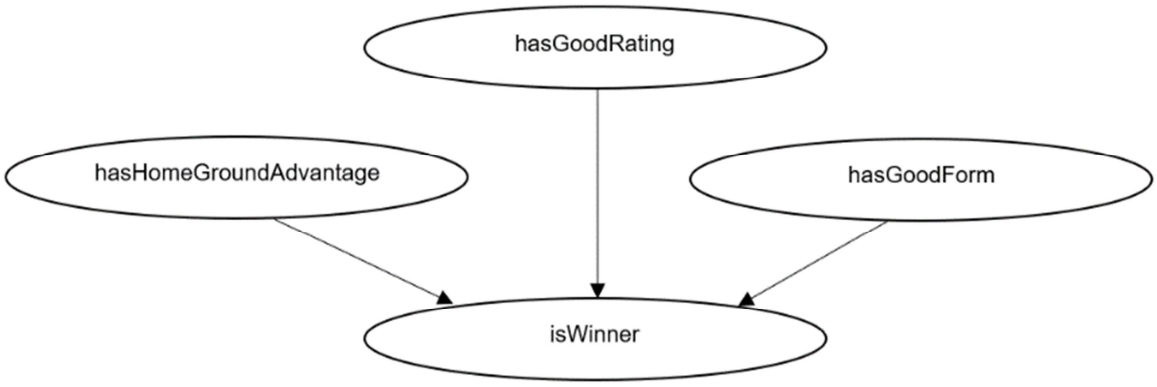

Figure 2. *K4F Dependency model.*

Next, the functional form of the dependencies is determined. In Figaro, the variable class constructors are used to express functional forms. According to Figure 2, the *is Winner* element is dependent on other elements, thus its functional form is:

$$\alpha = P(A) \tag{1}$$

$$isWinner = Flip(\alpha) \tag{2}$$

Flip is a construct used in Figaro to denote a Boolean value, and α is the probability of a win by a football team.

Has Good Rating is defined as:

$$\beta = Flip\big(P(A)\big) \tag{3}$$

$$\gamma = Flip\big(P(B)\big) \tag{4}$$

$$hasGoodRating = (\delta \rightarrow \gamma) \wedge (\neg\delta \rightarrow \beta) \tag{5}$$

β represents bad rating probability, γ represents good rating probability, and δ represents a win probability.

The functional form of *has Good Form* is:

$$\varepsilon = Flip\big(P(A)\big) \tag{6}$$

$$\zeta = Flip\big(P(B)\big) \tag{7}$$

$$hasGoodForm = (\delta \rightarrow \zeta) \wedge (\neg\delta \rightarrow \varepsilon) \tag{8}$$

ε is the probability of a team's bad form, ζ is the probability of a team's good form, and δ is the probability of a win.

Has Home Ground Advantage is defined as:

$$\theta = Flip\big(P(A)\big) \tag{9}$$

$$\vartheta = Flip\big(P(B)\big) \tag{10}$$

$$\text{Has Home Ground Advantage} = (\delta \rightarrow \vartheta) \wedge (\neg\delta \rightarrow \theta) \tag{11}$$

θ is the home ground loss probability, ϑ is the home ground win probability, and δ is the probability of a win.

The elements, their relationships, the functional form of the relationships and the numerical parameters together form a complete Figaro model for K4F.

This case study uses the expectation maximization (EM) learning algorithm and the variable elimination inference algorithm. Both algorithms are provided by Figaro.

6. Evaluation

It is necessary to evaluate an artefact thus providing insight on the effectiveness and quality of the artefact [72, 73]. K4F was evaluated using experimental method.

This experiment used Manchester United and Chelsea EPL teams. There previous games for 2017/2018 and 2018/2019 season were used in this experiment.

6.1. Learning Computation Results

Learning computation was carried out three (3) times on the batch module of the server component, corresponding to the intake of data. On the real-time module, learning was done as many times as new data was ingested into K4F. Learning computation was repeated at least five (5) times on both batch and real-time module to access duration.

Table 1. *First run learning duration in seconds.*

Learning time for Manchester United (s)	Learning time for Chelsea (s)	Total Learning Time (s)
0.48	0.994	1.474
0.416	0.537	0.993
0.383	0.534	0.917
0.503	0.878	1.381
0.581	0.757	1.338
Average learning time (s)		1.2126

Table 2. *Second run learning duration in seconds (batch module).*

Learning time for Manchester United (s)	Learning time for Chelsea (s)	Total Learning Time (s)
1.353	1.204	2.557
1.605	1.997	3.602
1.784	1.615	3.399
1.689	4.985	3.674
1.889	1.483	3.372
Average learning time (s)		3.3208

Table 3. *Third run learning duration in seconds (batch module).*

Learning time for Manchester United (s)	Learning time for Chelsea (s)	Total Learning Time (s)
2.948	3.222	6.17
3.152	3.22	6.372
2.625	2.942	5.567
2.577	2.687	5.264
3.047	2.587	5.634
Average learning time (s)		5.8014

On the first ingestion of data, learning on both batch and real-time module took approximately 1.2 seconds (See Table 1). Subsequently, as the size of data in the master database increases, the time to complete learning on the batch module also increased (See Tables 2 and 3). However, learning time on the real-time module remained in the same neighborhood.

6.2. Reasoning Computation Results

In K4F, a reasoning computation request is on the *is Winner* variable. K4F exposes reasoning on the batch module, server module and a combination of both. Table 4 shows the reasoning times in seconds.

Table 4. *Reasoning duration in seconds.*

	Reasoning time in real-time module (s)	Reasoning time in batch module (s)	Reasoning time in real-time & batch modules (s)
First Run	0.038	0.032	0.053
Second Run	0.043	0.031	0.06
Third Run	0.04	0.041	0.061
Fourth Run	0.029	0.030	0.063
Fifth Run	0.054	0.022	0.058
Average time (s)	0.0408	0.312	0.059

7. Conclusion

This paper presents a framework called Kognitor that proposes the adoption of probabilistic programming in big data processing. Kognitor also enables cost-effective and low latency data processing using Lambda architecture.

This paper started with a discussion on the background knowledge around big data processing and an analysis of related works. Then, the introduction of the framework, as well as an implementation to showcase effectiveness. Evaluation of Kognitor was presented using experimental method on a case study (K4F). Performance result from this evaluation shows low latency in data computation.

The aim of this paper is on probabilistic programming in big data computation. Thus, less effort was directed toward other components such as UX. This may constitute part of a future work. Another area for future work would be further evaluation of Kognitor framework using other evaluation methods.

References

[1] A. McAfee, E. Brynjolfsson, Big data: the management revolution., Harv. Bus. Rev. 90 (2012) 59–68. https://doi.org/10.1007/s12599-013-0249-5.

[2] D. Laney, 3D Data Managment: Controlling Data Volume, Velocity and Variety, Meta Group, 2001.

[3] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, C. Shahabi, Big data and its technical challenges, Commun. ACM. 57 (2014) 86–94. https://doi.org/10.1145/2611567.

[4] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, A. H. Byers, Big data: The next frontier for innovation, competition, and productivity, McKinsey & Company, 2011.

[5] T. Kraska, A. Talwalkar, J. Duchi, R. Griffith, M. Franklin, M. Jordan, MLbase: A Distributed Machine-learning System, 6th Bienn. Conf. Innov. Data Syst. Res. (2013).

[6] P. Szolovits, S. G. Pauker, Categorical and probabilistic reasoning in medical diagnosis, Artif. Intell. 11 (1978) 115–144. https://doi.org/10.1016/0004-3702(78)90014-0.

[7] R. Haenni, Towards a unifying theory of logical and probabilistic reasoning, Isipta. 5 (2005) 1.

[8] G. Luger, C. Chakrabarti, Knowledge-Based Probabilistic Reasoning from Expert Systems to Graphical Models, Handb. Probab. Theory Appl. (2008) 2–22. http://www.cs.unm.edu/~luger/23-Luger-Chakrabarti.pdf.

[9] N. Alon, Paul Erdős and probabilistic reasoning, in: Bolyai Soc. Math. Stud., 2013: pp. 11–33. https://doi.org/10.1007/978-3-642-39286-3_1.

[10] J. Gonzalez, Parallel and Distributed Systems for Probabilistic Reasoning, Carnegie Mellon University, 2012.

[11] C. Dobre, F. Xhafa, Parallel Programming Paradigms and Frameworks in Big Data Era, Int. J. Parallel Program. 42 (2014) 710–738. https://doi.org/10.1007/s10766-013-0272-7.

[12] Z. Ghahramani, Probabilistic machine learning and artificial intelligence, Nature. 521 (2015) 452–459. https://doi.org/10.1038/nature14541.

[13] L. A. Zadeh, Toward a perception-based theory of probabilistic reasoning with imprecise probabilities, in: Intell. Syst. Inf. Process., Elsevier, 2003: pp. 3–34. https://doi.org/10.1016/B978-044451379-3/50001-7.

[14] A. Pfeffer, Practical probabilistic programming, Manning, New York, 2016.

[15] S. Liu, A. H. B. Duffy, R. I. Whitfield, I. M. Boyle, Integration of decision support systems to improve decision support performance, Knowl. Inf. Syst. 22 (2010) 261–286. https://doi.org/10.1007/s10115-009-0192-4.

[16] L. R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE. 77 (1989) 257–286. https://doi.org/10.1109/5.18626.

[17] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological sequence analysis, Cambridge University Press, New York, 1998. https://doi.org/10.1017/CBO9780511790492.

[18] C. D. Manning, P. Raghavan, An Introduction to Information Retrieval, in: Online, 2009: p. 1. https://doi.org/10.1109/LPT.2009.2020494.

[19] L. De Raedt, K. Kersting, Probabilistic logic learning, ACM SIGKDD Explor. Newsl. 5 (2003) 31. https://doi.org/10.1145/959242.959247.

[20] E. Ábrahám, K. Havelund, Some recent advances in automated analysis, Int. J. Softw. Tools Technol. Transf. 18 (2016) 121–128. https://doi.org/10.1007/s10009-015-0403-0.

[21] D. Williams, Predictive coding and thought, Synthese. (2018). https://doi.org/10.1007/s11229-018-1768-x.

[22] Q. Zhang, C. Dong, Y. Cui, Z. Yang, Dynamic uncertain causality graph for knowledge representation and probabilistic reasoning: Statistics base, matrix, and application, IEEE Trans. Neural Networks Learn. Syst. 25 (2014) 645–663. https://doi.org/10.1109/TNNLS.2013.2279320.

[23] A. Pfeffer, Figaro: An object-oriented probabilistic programming language, 2009. http://www.cs.tufts.edu/~nr/cs257/archive/avi-pfeffer/figaro.pdf%5Cnpapers2://publication/uuid/0E83E526-451F-41EA-ACBE-7150FF7584D4.

[24] A. Sampson, Probabilistic Programming, (2015). http://adriansampson.net/doc/ppl.html (accessed March 25, 2018).

[25] D. Roy, Probabilistic Programming, (2018). http://www.probabilistic-programming.org/wiki/Home (accessed March 25, 2018).

[26] N. D. Goodman, A. Stuhlmüller, the Design and Implementation of Probabilistic Programming Languages, (2014). http://dippl.org (accessed March 25, 2018).

[27] M. Hicks, What is probabilistic programming? (The Programming Languages Enthusiast), (2014). http://www.pl-enthusiast.net/2014/09/08/probabilistic-programming/ (accessed March 25, 2018).

[28] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, S. J. Gershman, Building machines that learn and think like people, Behav. Brain Sci. 40 (2017) 72. https://doi.org/10.1017/S0140525X16001837.

[29] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent, HPCTOOLKIT: Tools for performance analysis of optimized parallel programs, Concurr. Comput. Pract. Exp. 22 (2010) 685–701. https://doi.org/10.1002/cpe.

[30] S. Shahrivari, Beyond Batch Processing: Towards Real-Time and Streaming Big Data, Computers. 3 (2014) 117–129. https://doi.org/10.3390/computers3040117.

[31] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, B. Moon, Parallel data processing with MapReduce, ACM SIGMOD Rec. 40 (2012) 11–20. https://doi.org/10.1145/2094114.2094118.

[32] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, Wei Ding, Data mining with big data, IEEE Trans. Knowl. Data Eng. 26 (2014) 97–107. https://doi.org/10.1109/TKDE.2013.109.

[33] S. Chen, W. Li, M. Li, X. Zhang, Y. Min, Latest Progress and Infrastructure Innovations of Big Data Technology, in: 2014 Int. Conf. Cloud Comput. Big Data, IEEE, 2014: pp. 8–15. https://doi.org/10.1109/CCBD.2014.25.

[34] W. Raghupathi, V. Raghupathi, Big data analytics in healthcare: promise and potential, Heal. Inf. Sci. Syst. 2 (2014) 3. https://doi.org/10.1186/2047-2501-2-3.

[35] J. Lin, F. Leu, Y. Chen, ReHRS: A Hybrid Redundant System for Improving MapReduce Reliability and Availability, in: 2015: pp. 187–209. https://doi.org/10.1007/978-3-319-09177-8_8.

[36] I. Taxidou, P. Fischer, Realtime analysis of information diffusion in social media, Proc. VLDB Endow. 6 (2013) 1416–1421. https://doi.org/10.14778/2536274.2536328.

[37] S. Sagiroglu, D. Sinanc, Big data: A review, in: 2013 Int. Conf. Collab. Technol. Syst., IEEE, 2013: pp. 42–47. https://doi.org/10.1109/CTS.2013.6567202.

[38] Y. Wu, L. Zheng, B. Heilig, G. R. Gao, Design and Evaluation of a Novel Dataflow Based Bigdata Solution, Proc. Sixth Int. Work. Program. Model. Appl. Multicores Manycores. (2015) 40–48. https://doi.org/10.1145/2712386.2712397.

[39] A. Vakali, P. Korosoglou, P. Daoglou, A multi-layer software architecture framework for adaptive real-time analytics, in: 2016 IEEE Int. Conf. Big Data (Big Data), IEEE, 2016: pp. 2425–2430. https://doi.org/10.1109/BigData.2016.7840878.

[40] S. K. Mohapatra, P. K. Sahoo, S.-L. Wu, Big data analytic architecture for intruder detection in heterogeneous wireless sensor networks, J. Netw. Comput. Appl. 66 (2016) 236–249. https://doi.org/10.1016/j.jnca.2016.03.004.

[41] S. Perera, S. Suhothayan, Solution patterns for realtime streaming analytics, in: Proc. 9th ACM Int. Conf. Distrib. Event-Based Syst. - DEBS '15, ACM Press, New York, New York, USA, 2015: pp. 247–255. https://doi.org/10.1145/2675743.2774214.

[42] M. Wang, J. Liu, W. Zhou, Design and Implementation of a High-Performance Stream-Oriented Big Data Processing System, in: 2016 8th Int. Conf. Intell. Human-Machine Syst. Cybern., IEEE, 2016: pp. 363–368. https://doi.org/10.1109/IHMSC.2016.64.

[43] M. Hirzel, S. Schneider, B. Gedik, SPL: An Extensible Language for Distributed Stream Processing, ACM Trans. Program. Lang. Syst. 39 (2017) 1–39. https://doi.org/10.1145/3039207.

[44] Apache Software Foundation, Apache Storm, (2015). http://storm.apache.org/ (accessed February 13, 2018).

[45] M. Zaharia, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, Apache Spark: a unified engine for big data processing, Commun. ACM. 59 (2016) 56–65. https://doi.org/10.1145/2934664.

[46] SQLstream, SQLstream - A SQL-based Real-time Stream Analytics Platform -, (2017). http://sqlstream.com/ (accessed February 15, 2018).

[47] B. Twardowski, D. Ryzko, Multi-agent Architecture for Real-Time Big Data Processing, in: 2014 IEEE/WIC/ACM Int. Jt. Conf. Web Intell. Intell. Agent Technol., IEEE, 2014: pp. 333–337. https://doi.org/10.1109/WI-IAT.2014.185.

[48] M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja, Lambda architecture for cost-effective batch and speed big data processing, in: 2015 IEEE Int. Conf. Big Data (Big Data), IEEE, 2015: pp. 2785–2792. https://doi.org/10.1109/BigData.2015.7364082.

[49] N. Marz, J. Warren, Big Data: Principles and best practices of scalable real-time data systems, Manning, New York, 2015. http://nathanmarz.com/about/.

[50] J. Kreps, Questioning the Lambda Architecture, O'Reilly. (2014) 1–10. https://www.oreilly.com/ideas/questioning-the-lambda-architecture (accessed October 18, 2017).

[51] N. D. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, J. B. Tenenbaum, Church: a language for generative models, in: Proc. 24th Conf. Uncertain. Artif. Intell., 2008: pp. 220–229. https://doi.org/10.1.1.151.7160.

[52] R. C. Fernandez, P. Pietzuch, J. Kreps, N. Narkhede, J. Rao, J. Koshy, D. Lin, C. Riccomini, G. Wang, Liquid: Unifying Nearline and Offline Big Data Integration, Conf. Innov. Data Syst. Res. (2015).

[53] Z. Hasani, M. Kon-Popovska, G. Velinov, Lambda Architecture for Real Time Big Data Analytic, ICT Innov. (2014) 133–143.

[54] V. Astakhov, M. Chayel, Lambda Architecture for Batch and Real- Time Processing on AWS with Spark Streaming and Spark SQL, (2015) 1–12. https://d0.awsstatic.com/whitepapers/lambda-architecure-on-for-batch-aws.pdf.

[55] M. Köhler, Y. Kaniovskyi, S. Benkner, Towards adaptive execution strategies for large-scale and real-time data analytics, Proc. Int. Conf. Parallel Distrib. Process. Tech. Appl. (2015) 447–454.

[56] G. Liu, W. Zhu, C. Saunders, F. Gao, Y. Yu, Real-time Complex Event Processing and Analytics for Smart Grid, Procedia Comput. Sci. 61 (2015) 113–119. https://doi.org/10.1016/j.procs.2015.09.169.

[57] J. C. C. Tseng, J. Gu, P. F. Wang, C. Chen, C. Li, V. S. Tseng, A scalable complex event analytical system with incremental episode mining over data streams, in: 2016 IEEE Congr. Evol. Comput., IEEE, 2016: pp. 648–655. https://doi.org/10.1109/CEC.2016.7743854.

[58] F. Yang, G. Merlino, N. Ray, X. Léauté, H. Gupta, E. Tschetter, The RADStack: Open Source Lambda Architecture for Interactive Analytics, in: Proc. 50th Hawaii Int. Conf. Syst. Sci., 2017: pp. 1703–1712. https://doi.org/10.24251/HICSS.2017.206.

[59] T. Yang, M. N. Shadlen, Probabilistic reasoning by neurons, Nature. 447 (2007) 1075–1080. https://doi.org/10.1038/nature05852.

[60] A. Tversky, D. Kahneman, Probabilistic Reasoning, Probabilistic Reason. 1131 (1983) 1124–1131. https://doi.org/10.1142/9789814291354_0006

[61] A. Prékopa, Probabilistic Programming, in: Handbooks Oper. Res. Manag. Sci., 2003: pp. 267–351. https://doi.org/10.1016/S0927-0507(03)10005-9.

[62] T. Gehr, S. Misailovic, M. Vechev, PSI: Exact Symbolic Inference for Probabilistic Programs, in: S. Chaudhuri, A. Farzan (Eds.), Int. Conf. Comput. Aided Verif., Springer, Cham, 2016: pp. 62–83. https://doi.org/10.1007/978-3-319-41528-4_4.

[63] F. Wood, J. W. van de Meent, V. Mansinghka, A New Approach to Probabilistic Programming Inference, in: 17th Int. Conf. Artif. Intell. Stat., Reykjavik, Iceland, 2014. http://arxiv.org/abs/1507.00996.

[64] A. Pfeffer, The Design and Implementation of IBAL: A General-Purpose Probabilistic Language, Introd. to Stat. Relational Learn. (2007) 34.

[65] B. Milch, B. Marthi, S. Russel, D. Sontag, D. L. Ong, A. Kolobov, Probabilistic models with unknown objects, Stat. Relational Learn. (2007) 352.

[66] T. Sato, A glimpse of symbolic-statistical modeling by PRISM, J. Intell. Inf. Syst. 31 (2008) 161–176. https://doi.org/10.1007/s10844-008-0062-7.

[67] A. Anikwue, B. Kabaso, Probabilistic Programming and Big Data, in: 2019 Int. Conf. Adv. Big Data, Comput. Data Commun. Syst., IEEE, 2019: pp. 1–6. https://doi.org/10.1109/ICABCD.2019.8851053.

[68] Z. Zhao, J. Pei, E. Lo, K. Q. Zhu, C. Liu, InferSpark: Statistical Inference at Scale, (2017). http://arxiv.org/abs/1707.02047.

[69] Lightbend Inc., Introduction - Akka Documentation, (2019). https://doc.akka.io/docs/akka/current/stream/stream-introduction.html (accessed June 8, 2019).

[70] Lightbend Inc, Akka, Actor-based message-driven runtime | @lightbend, (2010). https://www.lightbend.com/akka (accessed November 10, 2017).

[71] The Apache Software Foundation, Apache Cassandra Database, Cassandra. (2015). http://cassandra.apache.org/ (accessed December 20, 2018).

[72] A. R. Hevner, S. T. March, J. Park, S. Ram, Design Science in Information Systems Research, MIS Q. 28 (2004) 75–105. http://dblp.uni-trier.de/rec/bibtex/journals/misq/HevnerMPR04.

[73] A. Hevner, S. Chatterjee, Design Science Research in Information Systems, in: Des. Res. Inf. Syst., Springer US, Boston, MA, 2010: pp. 9–22. https://doi.org/10.1007/978-1-4419-5653-8_2.