ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 10, March 2016

Software for Big Data Analytics using Multi hop Processing

R.Ramesh , V.Geetha , T.Elakkiya /CSE,Sasurie College of Engineering
Guidedby: D.Prabhakaran (AP/CSE)

Abstract

Big data software needs to be delivered with quality, on time, and within budget. In existing system, the Single-hop processing of information flow has been implemented. In single-hop processing, data is processed by a single component and the output made readily available for immediate use. If this is the case, any inconsistencies in the results can be detected easier than in the multi-hop case. But in the proposed system, overcomes the issues of the existing system, Multi-hop processing of continuous information flow. In multi-hop processing, data is processed by multiple components before making the results available for use. If this is the case, any inconsistencies in the results would be more difficult to detect since it will not be obvious which component produced unreliable results. Also, any unreliable output in the multi-hop sequence will trickle through all subsequent processing. Additionally, the system enhances the efficiency using multi hop processing.

A survey of the literature reveals numerous attempts at defining big data, varying based on con- text, domain, and perspective. From the infrastructure's perspective, big data has been defined as data with high volume, velocity, and variety (3V), and unpredictability. In this context, it has also been defined as data with some aspect that's so large that cur- rent, typical methods can't be used to process it.[1,2] From the analytics' perspective, big data has been defined as data so large that it contains significant low probability events that would be absent from traditional statistical sampling methods.[3] From the business user's perspective, big data represents opportunities for gaining a competitive ad- vantage by gaining actionable intelligence.[4]

Each of these definitions provides descriptive and important aspects that must be sup- ported by big data software. Borrowing from these definitions, we propose a definition for big data software as "software that supports the time-constrained processing of continuous information flows to provide actionable intelligence."

The phrase software that supports acknowledges that big data software includes both infrastructure and analytics software these have been referred as big throughput and big analytics software, respectively.[5] Infrastructure software is needed to store, retrieve, transmit, and process big data. While it's essential to developing big data soft- ware, much of the emphasis and hype has been placed on the analytics portion of big data software. Nonetheless, our definition of big data software encompasses both types of software. The term time-constrained denotes the urgency in providing solutions. In a way, big data software shares a similar property with real-time software: late responses are wrong responses. The phrase *continuous* in- formation flows generalizes the input of big data software, which has the unique proper- ties of volume, velocity, and variety. This generalization also extends to other important information properties of big data input, such as continuity (data in motion versus data at rest). Data in motion (or data streams) can include potentially infinite data streams, at high speeds of arrival, with data whose density changes over time. Finally, actionable intelligence generalizes the output of big data software—that is, big data soft- ware exists to provide information that's

All Rights Reserved © 2016 IJARBEST

directly useful for immediate (strategic, operational, or tactical) use without having to go through the full intelligence production process.

Actionable Intelligence:

Actionable intelligence means much more than simply finding or summarizing data; it stresses the discovery of hidden (unknown and hard to find) patterns that can be used to predict concepts, events, trends, opinions, and so on to support decision makers. The expected use of actionable intelligence varies from system to system, with each variation increasing the complexity involved when engineering such systems. For this reason, it's important to derive a taxonomy for understanding actionable intelligence and for thinking about how it affects the complexity and development of big data software. The following three levels of actionable intelligence are proposed:

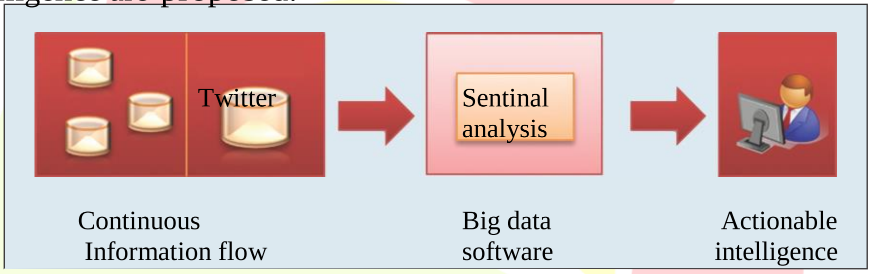

Figure 1. Single-hop processing of information flow. In single-hop processing, data is processed by a single component and the output made readily available for immediate use. If this is the case, any inconsistencies in the results can be detected easier than in the multi-hop case.

- Level 1 (L1), supervised actionable intelligence;
- Level 2 (L2), semi supervised actionable intelligence; and
- Level 3 (L3), unsupervised action- able intelligence.

In L1, the system serves as decision support for humans, so that humans supervise the intelligence produced by machines before taking action. In L2, machines are re- lied upon to take actions based on self-produced intelligence; how- ever, these actions can be reversed or corrected by humans. For example, an L2 spam detector uses ma- chine learning software to predict whether email messages are spam. Once this determination is made, the machine takes action to send incoming emails to spam or inbox folders. Admittedly, the process isn't perfect, and human intervention is required when misclassification occurs. In L3, machines are fully relied upon to take actions without requiring human intervention or correction. Not many L3 systems exist today; some recommender systems operate at L3, but most operate at L2, for obvious reasons.

Another important concept that needs to be explained before examining the engineering of big data software involves the intelligence production process. For big data soft- ware systems, it occurs primarily in two ways: single- and multi-hop processing. In single-hop processing, data is processed directly by a single component, and from that processing, all required actionable intelligence is produced and prepared for its intended consumer (human or machine), as Figure 1 shows.

A much more difficult approach(from the engineering perspective) involves multi-hop processing, the output of one component is the input of an other. This web of

All Rights Reserved © 2016 IJARBEST

components may be necessary to extract the most informative actionable intelligence that can help answer questions such as who, what, when, where, and why. Figure 2 presents a conceptual view of big data software executing under multi-hop processing and capable of providing various level of actionable intelligence. Such systems are expected to process continuous flows of YouTube news videos, trans- form input via speech-to-text, and extract sources and topics of conversation via text mining. Concurrently, the system monitors Twitter data, employs topic extraction seeking topics of interests, and uses sentiment analysis to predict public opinion. At each hop, the system produces actionable intelligence that's consumed by other components, and at the end, aggregate and correlate data to pro- vide intelligence that can be used to determine who's saying what and the perceived public reaction to that event.

Big Data Analytics Software Engineering

Having discussed big data software; its input, and outputs, let's turn our attention to the main problems encountered when attempting to engineer big data software. This requires the ability to specify requirements, design and construct software to meets those requirements. The focus is placed mainly on the big analytics portion of big data software.

The requirements P construction of software.

Requirement specification is crucial to the successful construction of software. The requirements problem is better presented using a simple model for functional requirement's specification and verification. Consider the simplest form of requirement specification one that specifies a single, verifiable function that the system must provide:

$r_i \leftarrow f_i(x_i)$,

where the requirement r_i is a function of some functional future x_i that the system needs to provide. Systems specified with such requirements are easily verified, simply by enumerating all requirement function evaluates to 1. That is assuming a vector x of the n functional features that the software needs to provide, a single requirement can be specified for each feature I, and verification of the software can be modeled using the equation below, where V=1 denotes successful verification:

$$V = \prod_{i=1}^{n} f_i(x_i).$$

Although ideal and simple, the model V for verification is inadequate for modeling other, more demanding software applications. For example, consider the case of real-time soft- ware. In these cases, software is not only required to provide a particular functional feature x_i, but it must do so within specified time-constraint t_j. Requirements for such software are functions of both x_i and t_j, as seen below:

$$r_i \leftarrow f_i(x_i \wedge t_i).$$

In these cases, r_i evaluates to 1 only when feature x_i has been met within t_i time. Thus, the proposed model V for verification needs to be ex- tended to account for these types of requirements:

All Rights Reserved © 2016 IJARBEST

$$\dot{V}_{real\text{-}time} = \prod_{i}^{n} f_i(x_i) \times \prod_{j}^{m} f_j(x_j, t_j).$$

Because big data software shares the same time-constrained properties as real-time software, time-constrained requirements also apply to big data software. At first glance, it would seem that *Vreal-time* is appropriate for verifying big data software. How- ever, both specification and verification of big data analytics software are far more troublesome because a major function expected from such systems involves making predictions on data streams. Every big data software product exists to process data that's dynamic, fast-changing, and most crucially, subject to concept drifts.[6] In big analytics, the term *concept drift* refers to changes in statistical properties of the concept to be learned. This phenomenon applies to a host of stochastic learning models—graphical models, Monte Carlo sampling methodologies, naïve Bayesian classifiers, and so on[7] —whose learned model parameters are directly correlated to the data stream used for their estimation. These changes occur over time in unforeseen ways, thus causing precision to become less accurate as time passes. This means that big analytics software could be specified to meet a specific accuracy threshold(using a given test dataset) and verified over some period p.

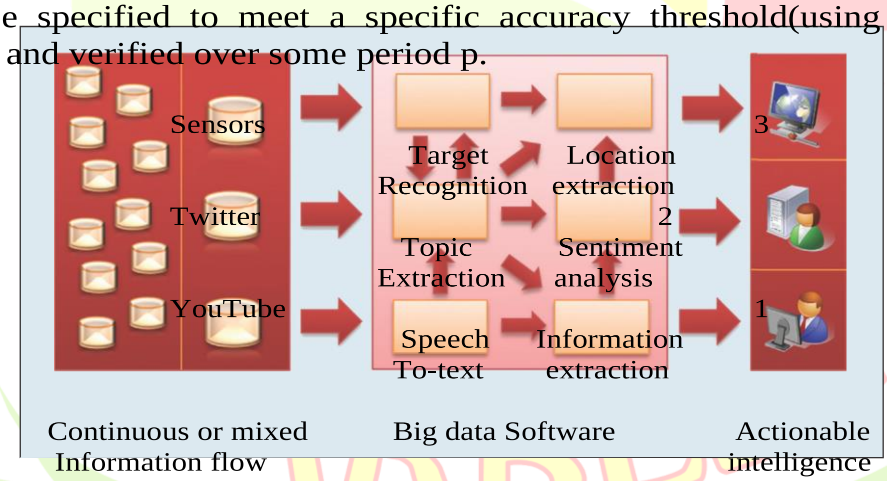

Figure 2. Multi-hop processing of continuous information flow. In multi-hop processing, data is processed by multiple components before making the results available for use. If this is the case, any inconsistencies in the results would be more difficult to detect since it will not be obvious which component produced unreliable results. Also, any unreliable output in the multi-hop sequence will trickle through all subsequent processing.

However, due to concept drift, the software's performance could begin to degrade over time without ever changing the software or hardware thus invalidating any previously verified analytics requirement! This expiration period could be a month, a year, or 10 years later. Clearly, Vreal-time doesn't capture this property of big data software. If software engineers are contractually bound to requirements, and requirements are bound to verification, how can software engineers write verifiable requirements when faced with concept drifts? To capture this properties of big data software, the requirements and verification model needs to be extended further:

$r_i \leftarrow f_i\ (x_i \wedge t_i \wedge p_i)$,
which mandates that the system provides a functional features xi, within some time constraint ti, and verifiable during some period pi, This new form of requirement is used to extend $V_{real\text{-}time}$ to accurately model the specification and verification of big data analytics software:

$$V_{BigData} = \prod_i f_i(x_i) \times \prod_j f_j(x_j, t_j) \times \prod_c f_c(x_c, t_c, p_c).$$

The model $V_{BigData}$ accounts for functional, time-constrained, and concept-drift-aware requirements. Concept evolution,[9] the emergence of new classes, is also another data dynamic that arises in rapidly evolving big data streams. Several algorithms[10] address concept drift and concept evolution; however, our concern is with the requirements and verification modeling of the software that could support these functionalities, not with the peculiarities of any one specific methodology. If the analytics software is required to incorporate the detection of new classes,
the proposed $V_{BigData}$ verification model captures this desire inherently as this would be yet another time- or periodically-constrained functional requirement.

Although the addition of p may seem trivial, it has significant engineering implications. The most notable one is that requirements specified for big data analytics software may (at most) result in systems that can only be verified reliably during some period p_i. Therefore, requirements for such systems need to be expressed in a way that they can be monitored automatically against software system events,[11] such as concept drifts. A suggested approach involves the use of satisfaction arguments, where the verifiability of requirements can only be argued after assuming some- thing about the domain.[11] These new forms of requirements might result in additional software that needs to be integrated into the system, which could result in increased cost and ex- tended schedules, since most likely they will have significant implications in the design, construction, testing, and maintenance phases. The addition of p also requires a paradigm shift in the way software contracts are thought about. Companies contracted to develop big data analytics software (such as US Department of Defense contractors) will have to deal with these issues in some way, especially as big data software continues to find its way into mainstream applications. These could possibly include application of big data software to national security, government policymaking, or safety-critical applications, in which case, other legal issues addressed by professional licensure would have to be taken into consideration.

The Design Problem

A glance at the literature points to certain key quality properties that are often cited for big data software, including usability, performance, re- liability, availability, security, interoperability, scalability, and testability, among others. Discussing all of these in the right amount of detail requires far more space than the one allotted for this article. For this reason, focus is placed on the reliability and secu- rity of big data software.

Reliability. Perhaps the most problematic quality attribute to design in big data software is reliability. The problem lies mainly in the software's inability to determine (with absolute certainty) correct outcomes. Consider the sentiment analysis example presented earlier, where big data soft- ware monitors Twitter. In this case, input is data streams in the form of noisy Twitter messages, and output is actionable intelligence in the form of predicted (positive, neutral, or negative) sentiment. At best, the system provides the best-effort prediction of sentiment. This prediction could be highly accurate, but minimal or no guarantees are made about the reliability of such prediction in some cases, this would even be difficult for humans. Under single-hop, L1 and L2 actionable intelligence conditions, verifying the reliability is difficult, but not impossible, since humans can use judgment and domain knowledge to visually detect inaccurate or extreme results.

However, under multi-hop, L3 actionable intelligence conditions, the state of the art is insufficient for equipping software with such functionality. The question then arises, how do software engineers design for reliability in big data software? The design problem is related to the specification problem discussed earlier and is an open area of research in software engineering design. Suggested solutions include discovering new (or adapting old) architectural tactics to help increase the reliability of such systems. For example, architectural tactics that exploit deterministic metadata can be incorporated to help concurrently gauge the reliability of predictions. To deal with concept drift, a concept-drift monitor or change detection component[6] could be incorporated into the software de- sign to detect and alert users when analytic models are no longer reliable. However, achieving this functionality is no easy to take because the "design of a change detector is a compromise between detecting true changes and avoiding false alarms. Other works have proposed the inclusion of similar monitors to detect at run time whether other system features comply with its requirements during its lifetime. Another suggested approach involves extending the concept of metamorphic properties[13] for the creation of monitors capable of detecting misbehavior using these properties, essentially crosscutting the requirements, design, and testing phases. These approaches (and others) need to be explored further to determine their adequacy for providing efficient solutions to these complex problems.

Security. In big data software, perfectly functioning software can by itself be its own security vulnerability, such as is the case in adversarial machine learning.[14] Two types of security attacks that exist for big data software involve execution phase attacks and training phase attacks. In the former, data streams are tapped to influence the actionable intelligence generated by the big data soft- ware. Consider the case of important organizations (such as government) using big data software to gauge public opinion from social media for a particular topic. In such a system, malicious attackers can create data generators that influence big data software to produce unreliable or malicious results—this is done with- out ever having to break software, find backdoors, and so on. In the case of Twitter sentiment analysis, attackers can simply create a public account to inject pollutant datasets.

In training phase attacks, attackers gain access to user accounts and inject the big data software with an adversarial training dataset designed to influence the machine learning training process to make the classifier misbehave.[14] This would defeat the purpose of most recommender systems and could lead to great financial loss for companies. To prevent these attacks , advanced endpoint input validation /filtering techniques[15] must be developed to assure data integrity and authenticity while (in some cases) preserving data privacy. Another particularly important problem to some big data software is denial of service (DoS). In most traditional applications, denying access to systems can result in nuisances for users and lead to significant costs to organizations. In big data software, denying access to data streams for some time

All Rights Reserved © 2016 IJARBEST

period *q* can render the whole system a failure because they might have been designed to work only at *q*. Consider emergency evacuation big data software that relies on social media data streams to enhance traffic routing, emergency dispatching, and so on during emergencies.[16] A DoS at- tack on the data source—instead of the target big data application—at period *q* could have catastrophic results, such as is the case in safety- critical applications. This, of course, could influence many aspects of the software engineering process. Other important security concerns include securing computations in distributed programming frameworks[15] and devising scalable privacy-preserving data mining computations.[15,17] For these and other security concerns, re- search is needed to better understand the attack surface of big data applications, to identify use (and misuse) cases, and to devise threat models for each possible big data attack.[15]

The Construction Problem

The construction problem is mainly driven by big data software requiring more than just programming skills. Today, most learning algorithms are created by machine learning scientists, who aren't necessarily the best programmers.[18] Specific issues mentioned in the literature include sloppiness, maintainability, debugging, and reproducibility.[18] Most of these problems have been addressed within the software engineering community,
where coding styles, peer reviews, configuration management software, and other tools are employed. How- ever, a common trend exists where many machine learning advances are being used for improving software engineering, but not enough of soft- ware engineering advances are being used for improving machine learning software.[13]

Other construction issues include lack of tools and frameworks that support assistive development.[19] Popular development applications such as Waikato Environment for Knowledge Analysis (WEKA), Excel, Octave, or R are designed for machine learning software but not big data software. Applications of this sort require all the data to reside in memory, which isn't the case for big data. Some tools are beginning to exist, such as the Massive Online Analysis (MOA) framework for data streaming,[8] but more are needed. Construction for big data software is also particularly complex because the inherent nature of big data requires multiprocessing technologies, such as CUDA, GPUs, MapReduce, Hadoop, and so on, which must be mastered during the construction phase. Recently, new software frameworks, specifically targeted at improving the usability of these technologies, are striving to re- duce the learning curve for big data software engineering. Examples include Mahout, which leverages the Hadoop processing environment for large-scale machine learning; Spark, built on the Scala language; and Storm for real-time analysis. Undoubtedly, by the time of publication, a few more might have arrived.

Although not the focus of our present effort, it's worth mentioning that efficient big data analytics also requires you to keep in mind the distributed data store and retrieval technologies underlying the analytics computation. These technologies and frameworks include NoSql (sup- ported in Cassandra and Hypertable) and SciDB. Under this massive and fast-changing technology landscape, developers not only have to keep up with the state of the art, but they must also expend significant development effort in experimenting with different algorithms, multiprocessing techniques, and software frameworks.[19]

This, in turn, will strongly impact cost and schedules. Furthermore, the variety of big data may often entail multimedia processing (video, speech, text, and so on) for a single project, which increases complexity. Data cleansing and preparation efforts also add significant overhead during construction.[20] Future work should ad- dress these issues, together with the creation of out-of-the-box solutions and exploratory tools to help during both training as well as executing big data software.[2 ,21]

The Testing Problem

Verifying the results of massive data stream processing is impractical for humans, if not impossible—it's the reason why big data software exists in the first place. Throughout this process, many defects can go unnoticed. The research literature reports an "extreme imbalance between the number of published algorithms versus those really workable in the business environment," and reasons given for this imbalance include academic researchers not taking into account business environments.[21] In other cases, transferring from algorithm development to implementation can be complex, leading to problems and defects being injected along the way. Specific concerns mentioned in the literature include untrustworthy data, bad assumptions, and incorrect data scientists ignoring important details to promote new methods and helping their new algorithms along by making sure that nothing goes wrong during the application of a method.[18]

For these reasons, the ability of non- data scientists to test the results of big data software is crucial.

Testing of big data software is problematic because these programs belong to the class of non-testable software, where no test oracles exist for verification. A proven approach for testing such systems is metamorphic testing,[13] which requires the discovery of metamorphic properties (MP) that can be used to test the validity of machine learning software. Consider the following metamorphic properties proposed for the Twitter sentiment analysis example. Based on domain knowledge, a vocabulary v is identified for words associated with positive sentiment, where v_i represents a single positive word in a tweet:

$v = [v_1, v_2, \ldots, v_n]$.

These verification techniques should be explored; if successful, they could be employed by test specialists without them ever needing to have expertise in machine learning. Open areas of research in this area involves the discovery of domain-specific MP in various big data software, which requires deep understanding of ma- chine learning algorithms, software testing, and the domain in question. These discoveries would allow testers to transition to more appropriate in vivo testing approaches and will also provide significant crosscutting contributions for both requirements and design of big data software, as discussed earlier. Can we really engineer big data software? Yes, we can. No one day's mainstream big data software, such as Facebook, Netflix, and Amazon. Surely, a great deal of successful software engineering work has taken ~v can be derived for capturing synonyms, antonyms and negations of v, respectively:

$□ = [v□_1, v□_2 ,\ldots, v□_n]$

$v = [v_1, v_2 ,\ldots v_n]$ – – – –

$\neg v = [\ \llbracket \neg v \rrbracket\ 1,\ \llbracket \neg v \rrbracket\ 2, \ldots,\ \llbracket \neg v \rrbracket\ n]$.

Twitter messages, such that the sentiment classification for messages using each v_i word should result in positive sentiment; reusing the same messages but swapping each v_i word with each $\bar{v}_i$ should also result in positive sentiment classification; and messages with place. Perhaps a more relevant question is, has the rest of the software engineering community mastered the engineering of such systems? We don't think so.

Today, engineering these systems still entails seeking the most accurate prediction while accepting solutions that sacrifice accuracy for the sake of not having a solution at all.[8] Moreover, requirements for the analytics portions of big data soft- ware—to quote Captain Barbosa from *Pirates of the Caribbean: The Curse of the Black Pearl*—are "more what you'd call 'guidelines' than ac- tual rules." Many of the issues presented so far, and more, still remain open to research. Software engineering research is needed to fully under- stand the role of requirements, design, and testing in big data software, and to characterize their crosscutting effects. Research work needs to take place to build architectures, tools, and frameworks;[23] to improve visualization and infrastructure software; to move from black-box to white-box algorithms and increase usability in adaptive learning systems;[6] and to understand the role of professional licensure and contract liability in big data software, especially as it makes its way to mainstream applications. In- deed, much work is needed to reach the point where high-quality big data software can be created on time and within budget.

References

1.Carlos E.Otero and Adrian Peter "Research Directions for Engineering Big Data Analytics Software", Florida Institute of Technology" 1541-1672/15/$31.00 @ 2015 IEEE published by the IEEE Computer Society.

2.T.Kraska "Finding the Needle in the Big Data Systems Haystack," IEEE internet Computing, vol. 17, no. 1, 2013, pp. 84-86.

3.J. Gama et al ., "A Survey on concept Drift Adaptation," ACM Computing Surveys, vol. 46, no. 4, 2014, article no.44.

4.A.Bifet et al., "MOA Massive online Analysis," 2014; http://heanet.dl.sourceforge.net/project/moadatastream/documentation/StreamMinning.pdf,

5.M.Masud et al., "Detecting Recurring and Novel Classes in Concept-Drifting Data Streams," Proc. 2011 IEEE Int'l Conf. Data Mining, 2011, pp. 1176-1181.

6.W.N. Robinson, "A Roadmap for Comprehensive Requirements Monitoring," Computer, vol. 43, no. 5,2010,pp. 64-72.

7.J.D. Tygar, "Adversarial Machine Learing," IEEE Internet Computing, vol. 15, no. 5, 2011, pp. 4-6.

8.J. Yin et al., "Using Social Media to Enhance Emergency Situation Awareness," IEEE Intelligent Systems, vol. 27, no. 6, 2012, pp.52-59.

9.F. Shull, "Getting an Intuition for Big Data," IEEE software, vol. 30, no. 4,2012, pp. 88-92.

10.D.E. O'Leary, "Artificial Intelligence and Big Data," IEEE Intelligent Systems, vol. 28, no. 2,2013, pp. 96-99,2013.

11.S.Soren, "The Need for Open Source Software in Machine Learning," J.Machine Learning, vol. 8, no. 10, 2007, pp.2443-2466.

All Rights Reserved © 2016 IJARBEST