THE DZONE GUIDE TO

BIG DATA PROCESSING

VOLUME III

BROUGHT TO YOU IN PARTNERSHIP WITH

dear reader,

Earlier this year Wired Magazine published an article on "The End of Code" with the subtitled explanation that training computers will soon replace programming them.

Well of course that isn't true. Every formal system needs a Zermelo and a Fraenkel to give the last β-reduction a back foot – and a Gödel to explain where it can never go.

But it is certainly true that an increasing number of problems can be solved without transparently representing the solution in formally verifiable business logic. It's a bit bizarre that artificial neural nets work as well as they do, as many (straggling) proponents of artificial *general* intelligence love to observe – but it's no more bizarre than the fact that layered action potential thresholds work in wetware as well as they do. Neuroscience doesn't have to be solved before we can recognize a thousand objects by pushing tensors through a gradient descent.

Nor do we need to construct a Byzantine generals scenario to run a MapReduce job on a Hadoop cluster, nor rediscover Hamming codes to run a logistic regression on a dozen machines coordinated by Spark. Truly fault-tolerant distributed computing is ridiculously hard and very far from most applications' actual business logic – so for 'big data' problems the lifting done by the platform/framework is even heavier than usual.

But as software developers we want to have a black box *and* be able to crack it open. Fortunately, in the relatively young space of too-big, too-fast, and too-heterogeneous data, techniques and toolchains are coupled even more tightly than normal. You can't just say 'split this job for me, magic Hadoop'; you also have to start thinking in map() and reduce(). You can't just tell TensorFlow 'find patterns in this stream'; you also need to understand which transformations to apply at exactly which node in the dataflow graph, and keep backpropagating manually until you appreciate how useful a pre-baked automatic differentiation engine can be.

So we've jam-packed the *2016 DZone Guide to Big Data Processing* with those two sides of the 'so much data!' coin – the 'problem' side (too big, too fast, too heterogeneous) and the 'magically I didn't program this in explicitly' side (machine learning) – so you can use these frameworks *and* unpack what they're doing under the hood. Read it and make a brain.

BY JOHN ESPOSITO
SENIOR RESEARCH ANALYST, DZONE
RESEARCH@DZONE.COM

TABLE OF CONTENTS

EDITORIAL

CAITLIN CANDELMO
DIRECTOR OF CONTENT + COMMUNITY

MATT WERNER
CONTENT + COMMUNITY MANAGER

MICHAEL THARRINGTON
CONTENT + COMMUNITY MANAGER

NICOLE WOLFE
CONTENT COORDINATOR

MIKE GATES
CONTENT COORDINATOR

INDUSTRY + VENDOR RELATIONS

JOHN ESPOSITO
SENIOR RESEARCH ANALYST

TOM SMITH
RESEARCH ANALYST

BUSINESS

RICK ROSS
CEO

MATT SCHMIDT
PRESIDENT & CTO

JESSE DAVIS
EVP & COO

KELLET ATKINSON
DIRECTOR OF MARKETING

MATT O'BRIAN
SALES@DZONE.COM
DIRECTOR OF BUSINESS DEVELOPMENT

ALEX CRAFTS
DIRECTOR OF MAJOR ACCOUNTS

JIM HOWARD
SR ACCOUNT EXECUTIVE

CHRIS BRUMFIELD
ACCOUNT MANAGER

PRODUCTION

CHRIS SMITH
DIRECTOR OF PRODUCTION

ANDRE POWELL
SENIOR PRODUCTION COORDINATOR

G. RYAN SPAIN
PRODUCTION PUBLICATIONS EDITOR

ART

ASHLEY SLATE
DESIGN DIRECTOR

SPECIAL THANKS to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this report a great success.

WANT YOUR SOLUTION TO BE FEATURED IN COMING GUIDES?
Please contact research@dzone.com for submission information.

LIKE TO CONTRIBUTE CONTENT TO COMING GUIDES?
Please contact research@dzone.com for consideration.

INTERESTED IN BECOMING A DZONE RESEARCH PARTNER?
Please contact sales@dzone.com for information.

Executive Summary

Question: who cares whether your data is "big"? Answer: anyone who has to say "no, that's impossible" to a business analyst because that dataset fills just too many GBs, or has to learn a new in-memory DBMS because of growing throughput requirements, or suddenly needs to internalize MapReduce because no machine is (cost-effectively) powerful enough to scale vertically anymore. Or, more positively, anyone who wants to build a recommendation engine, or run analysis on more dimensions than a query in SQL can conveniently handle, or forecast the fuzzy state of a complex system at some nontrivially future time.

MANY DEVELOPERS ARE WORKING WITH VOLUMES AND VELOCITIES OF DATA THAT PROBABLY DO REQUIRE CLUSTER PROCESSING

DATA: 22% of developers work with datasets larger than 1TB. 27% of developers build systems that are required to process more than 1000 messages per second. 37% of developers are planning to adopt a new "Big Data" technology over the next six months.

INTERPRETATION: 1TB is a crazy-huge amount of data: 15 million rows in a tiny, two-column (both bigints) table in a MySQL database (running InnoDB) fill barely 0.13% of a terabyte on disk. 1000 messages/second is insanely high throughput: more than a quarter of developers need to process messages in one millisecond each (or less). These numbers alone are not enough to determine how many jobs require multi-node processing, but they are high enough to counter suspicions that a negligibly small percent of applications require distributed data processing.

RECOMMENDATIONS: Consider whether you need to use a cluster computing technology like Hadoop or a rapid ingestion system like Kafka or NiFi. Technical and business motivations and constraints interact as always, but both actual volume and velocity numbers and planned "Big Data" technology adoption rates suggest that a significant portion of developers are working with data sets and speeds beyond what their current systems can handle. (To develop a more concrete understanding of MapReduce patterns, see Shital Katkar's article on page 20. For architectural requirements of large-scale analytics systems, see Rohit Dhall's article on page 22.)

SPARK CONTINUES TO GAIN GROUND ON HADOOP

DATA: Of developers who plan to adopt a new "Big Data" technology in the next six months, 19% plan to adopt Hadoop and 14% plan to adopt Spark.

INTERPRETATION: Hadoop (released in 2011) remains the go-to framework for large-scale data processing. This is presumably a function of the analytical utility of paired map() and reduce() functions, the versatility of the MapReduce paradigm, the beyond-MapReduce capability of YARN (released with Hadoop 2), and probably also the comparative maturity and familiarity of the Hadoop API. The much younger Spark (released in 2014) allows more flexible clustered workloads (for both discretized streaming and batch jobs) and unique distributed data structures that remain resiliently in memory for many iterations (which massively improves the performance of many machine learning algorithms). As a result of the latter, Spark is very memory-intensive; but the decreasing cost of DRAM and SSDs lowers the infrastructure barrier to Spark adoption.

RECOMMENDATIONS: Consider Spark where you think Hadoop might do—if you have the available system resources (especially memory) and a definite enough roadmap to predict a high probability of both batch and streaming workloads in the foreseeable future. (For an overview of the Spark ecosystem see Frank Evans' article on page 8 in this Guide.)

DEVELOPERS ARE PREPARED TO DEVELOP MACHINE LEARNING APPLICATIONS

DATA: More than half of developers are familiar with nine out of ten popular machine learning algorithms. (The exception is the apriori algorithm, but given that it is a very straightforward way to calculate association by overlap it seems plausible that many developers have implemented something similar under another name.) 80% are either familiar with neural nets but haven't used them (53%) or use them occasionally (18%) or use them regularly (10%). But for seven out of these ten algorithms, more developers are familiar with but have not used them than have used them even occasionally. (The exceptions are linear regression, k-means clustering, and decision trees.)

INTERPRETATION: Developers have an untapped ability to implement machine learning applications, especially using neural nets. This is probably a function of numerous factors such as domain immaturity (more applications would be improved by machine learning than enjoy resources dedicated to applying machine learning), toolkit immaturity (especially in language ecosystems other than R and Python, although this is rapidly changing), general media hype (especially around neural nets), and sheer mathematical "cool factor."

RECOMMENDATIONS: Move from general familiarity with machine learning algorithms to practical experience. Actively seek opportunities to apply machine learning to improve existing applications. Start with the powerful libraries and active communities available for machine learning in Python and R; then, for the JVM ecosystem, consider Weka, Java-ML, and Deeplearning4j. (For an introduction to neural nets, see Emmett Coin's article on page 12. For a whirlwind tour of machine learning algorithms and techniques, see Tamis Achilles van der Laan's article on page 28.)

Key Research Findings

1511 developers responded to our 2016 "Big Data" survey. The survey was distributed on the web at DZone.com, via social channels (Twitter, LinkedIn, Facebook), and via email to 215,000 addresses. Demographics are as follows:

- **83% of respondents' companies use Java; 63% use JavaScript; 28% use Python; 27% use C#**
- **66% of respondents' primary programming language is Java**
- **73% are developers or developer team leads; 6% are self-employed; 5% are C-level executives; 4% are business managers or analysts; 3% are in IT operations.**
- **63% have been IT professionals for 10 or more years; 37% have been IT professionals for 15 or more years**
- **30% of respondents are from North America; 42% are from Europe**

HOW BIG IS BIG DATA?

A. VOLUME

For a practicing developer, the 'big' in "Big Data" just means "makes me handle the data in a way significantly affected by its quantity along the dimensions of volume, velocity, variety, or whatever." (In other words, "pain" is the catalyst we used to operationalize the buzzword "big.") The first two of these quantities (volume and velocity), and the relation of those quantities to available single-node compute resources, is crucial to determining whether workload distribution across multiple nodes—with all the inconvenience and complexity that entails—is really necessary.

To get a better sense of how data volume is affecting processing technologies, we asked developers: "How big is your biggest data set (in GB)?" A bucketed distribution of results is visualized below and, in a different format, in the infographic on page 16.

Two observations are worth making immediately. First, three quarters of developers (78%) don't deal with datasets larger than 1TB. (The flip side—that nearly a quarter of developers do deal with 1TB+ datasets—is impressive in its own right.) Because many servers now ship (or can be spun up) with SSDs 1TB or larger (in fact, many are available to consumers for less than $300), and because many modern database management systems (such as Postgres) are extremely efficient at swapping between physical and virtual memory, it seems likely that a large majority of developers can scale vertically to process their largest datasets. Whether or not it makes sense to do is then a function of requirements other than max dataset size (response-time SLAs, cost to scale vertically vs. horizontally, architectural limitations imposed by distributedness, etc.). Moreover, we did not ask how frequently developers deal with datasets of this size (i.e. how batchable these datasets may be); further research is required to determine frequency and type of manipulation of 1TB+ datasets.

Second, more than a third of developers (35%) don't deal with datasets larger than 25GB, and just over half (54%) don't deal with datasets larger than 100GB. These sizes are far from trivial for many applications, of course—just run an outer join on two 5GB tables—but performance ceilings on datasets of this size are dominated by architectures, algorithms, and competition for shared resources, not physical limitations that require a dedicated processing cluster.

B. VELOCITY

Velocity can be measured along many dimensions, of course, but for our survey we chose the two most fundamental: latency and throughput (max required messages processed per second).

LATENCY: max milliseconds from data creation to required response

- Max acceptable latency is a function of both domain and technical requirements. Moreover, as architectures (e.g. microservices, or anything service-oriented) become more distributed, technical demands on response time become increasingly complex. It is therefore impressive but unsurprising that nearly half of respondents (48%) are at least sometimes required to meet <100ms response time SLAs. Note that in this survey we asked not for server response time to a client request but rather for max milliseconds from data creation to required response, irrespective of the nature of the response. (So, for example, both 'write to disk' and 'transmit an HTTP response' required within n milliseconds from data creation would be counted among all responses required within n.)

01 **HOW BIG IS YOUR BIGGEST DATASET (IN GBs)?**

MIN	MAX	PERCENT
0	25	**35%**
26	100	**19%**
101	500	**13%**
501	1000	**11%**
1,001	10,000	**14%**
10,001+		**8%**

02 **WHAT ARE YOUR FASTEST DATA PROCESSING REQUIREMENTS?**

MIN	MAX	PERCENT
0	25	**33%**
26	100	**15%**
101	500	**17%**
501	1000	**12%**
1,001	10,000	**18%**
10,001+		**5%**

- Slightly more surprising is the other end of the velocity spectrum: nearly a quarter of developers (23%) have no requirements to respond within less than 1 full second. This seems rather long for many "normal" types of response (e.g. render a dynamic webpage), but reasonable for others—for example, a SQL query run regularly to sum values and store results, or a many-dimensional tensor factorization for a collaborative filtering recommendation system—so another survey would be needed to discover an application-specific distribution of response types.

THROUGHPUT: max required messages processed per second

- Latency affects throughput directly, of course, but business requirements may specify latency and throughput separately, or one and not the other. Moreover, while (all other things being equal) an increase in latency results in linear decrease in throughput, an increase in throughput (holding available system resources constant) often results in bottlenecks that increase average latency much more rapidly than linearly. The result is that low latency is a good benchmark for application performance, but high throughput is more likely to cause the pain that distinguishes "big" data from "normal" data in practitioners' eyes.
- To understand developers' throughput requirements, we asked "How many messages is your application required to process per second (max)?" Results are visualized in the infographic on page 16, but a few points are worth making here. First, over a quarter (27%) of respondents are required to process more than 1000 messages per second. (In this survey we didn't ask for any specifics about the contents or format of these messages.) This means that, for a quarter of developers, each message must be processed in (on average) one millisecond or less. Slightly less than half of respondents, on the other hand, have no more than 100 messages to process per second—a throughput that, in many cases, may be handled serially.

WHICH DATA SOURCES AND TYPES GET TOO BIG, FAST, AND HETEROGENEOUS?

Dataset size, message throughput, and cluster size are content-indifferent numbers, and many data processing systems gain value precisely because they are not tightly coupled to specific data sources or types. But the distributability of any data processing job depends on the splittability of the work, which is determined by the interdependence of the different elements in the dataset, the interrelation of messages processed, and higher-level requirements determined by business constraints (e.g. how and where in your application to prioritize consistency, availability, or partition-tolerance; or whether higher-throughput I/O can be installed in your datacenter; etc.).

To understand the information content of the "big" data that developers are handling, we asked: which data sources and types cause you the most pain, along the three dimensions of volume ("too big"), velocity ("too fast") and variety ("too heterogeneous")? Results are summarized in graphs below.

HOW MANY NODES ARE BEING USED TO PROCESS THIS DATA?

If a single machine can process your data within the time limits required, then your work as a data wrangler—all other things being equal—is simpler than if your dataset can be processed only on multiple machines. Hadoop has been widely used for such multi-node processing jobs because of its ability to coordinate multiple nodes efficiently (using a simpler consensus algorithm than Paxos) and with a comparatively easy-to-use API—in spite of the learning curve required to divide work into map() and reduce() functions, and in spite of the limitations (particularly the need to complete workloads in batches) imposed by the MapReduce paradigm.

Consensus algorithms, however (and however clever), are notoriously tricky to scale. The number of nodes managed (by ZooKeeper, for example) should therefore be as small as possible

03 WHAT DATA SOURCES CAUSE YOU THE MOST PAIN?

TOO BIG:

1. Files (documents, media)
2. Server Logs
3. User-Generated Data (social, game, blog, etc.)
4. ERP, CRM, or other enterprise systems
5. Sensor Data or Remote Hardware
6. Supply Chain, Logistics, Other Procurement
7. Other

TOO FAST:

1. Sensor Data or Remote Hardware
2. Server Logs
3. ERP, CRM, or Other Enterprise Systems
4. User-Generated Data (social, game, blog, etc.)
5. Supply Chain, Logistics, Other Procurement
6. Files (documents, media)
7. Other

TOO HETEROGENEOUS:

1. Files (documents, media)
2. ERP, CRM, or other Enterprise Systems
3. User-Generated Data (social, game, blog, etc.)
4. Supply Chain, Logistics, Other Procurement
5. Server Logs
6. Sensor Data or Remote Hardware
7. Other

04 WHAT DATA TYPES CAUSE YOU THE MOST PAIN?

TOO BIG:

1. Relational (Flat Tables)
2. Audio and/or Video
3. Web Logs and Clickstreams
4. Complex (Graph, Hierarchical)
5. Event Data (Usually Real-Time)
6. Semi-Structured (JSON, XML)
7. Social Media (blogs, tweets, social network, etc.)

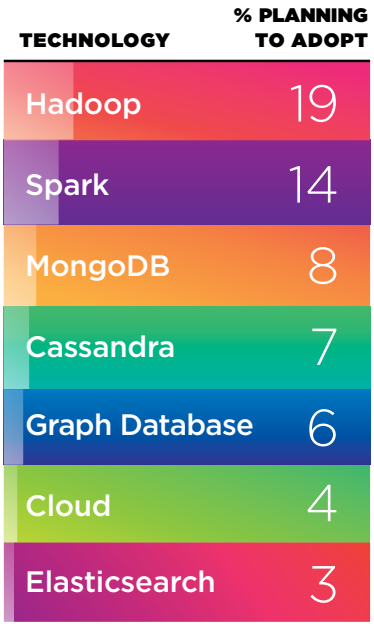
TOO FAST:

1. Event Data (Usually Real-Time)
2. Web Logs and Clickstreams
3. Semi-Structured (JSON, XML)
4. Complex (Graph, Hierarchical)
5. Social Media (blogs, tweets, social network, etc.)
6. Relational (Flat Tables)
7. Audio and/or Video

TOO HETEROGENEOUS:

1. Other
2. Complex (Graph, Hierarchical)
3. Semi-Structured (JSON, XML)
4. Social Media (blogs, tweets, social network, etc.)
5. Audio and/or Video
6. Relational (Flat Tables)
7. Web Logs and Clickstreams

05 WHAT NEW TECHNOLOGIES ARE YOU PLANNING TO ADOPT IN THE NEXT SIX MONTHS?



to achieve the task required (factoring out internal skills growth, as discussed above), and the number of nodes in existing data processing clusters may indicate that existing distributed workload management tools and algorithms are adequate or need to be modified to accommodate horizontal scale.

To learn how many nodes developers are using in their data processing clusters, we asked for (a) both max and typical cluster size, (b) max and typical amount of data processed by each cluster, and (c) how processing and persistence tasks are divided (either within a cluster or in separate clusters). The relation between dataset size and cluster size is visualized in the infographic on page 16. Analysis of the results of other questions we asked (about cluster size and organization) will be published in a future article on dzone.com.

PLANNED ADOPTION OF BIG DATA TOOLS

The business case for adopting Big Data technologies is more complex than whether or not data volume/velocity/variety physically requires a specialized algorithm or tool. So we asked respondents separately whether they were "planning to adopt new 'Big Data' technologies in the next six months." 37% of respondents reported that they are planning to do so.

We then asked those 37% (426 respondents out of 1154) what new technologies they were planning to adopt. The results show that, in spite of the explosion of non-batch tools and frameworks, Hadoop is still the go-to Big Data technology for new adopters—although Spark is getting close. It is also worth noting that three basic, mildly constrained NoSQL storage models are represented in this "top seven" list, with document-oriented (MongoDB, possibly Elasticsearch), column-(family-)oriented (Cassandra), and graph (Neo4j was the only particular graph DBMS mentioned more than once), with none clearly dominating (unless you count Elasticsearch as document-oriented in spite of its full-text indexing defaults).

MACHINE LEARNING ALGORITHM USAGE AND AWARENESS

The nebulous concept "data science" does not capture the specific nature of knowledge applied and work done to turn data into information (let alone insight or action or wisdom or whatever): for example, if you "do data science," then do you do ETL, or apply and tune machine learning algorithms, or visualize multidimensional data, or…? But even programmers without academic math training naturally contribute to work in "data science," simply because they understand how to look at unformed data and transform (or extract, or load…) it into something useful. Machine learning is one of the areas of data science that often comes naturally to developers, because many machine learning models are best described algorithmically (often in Python or pseudo-code).

To get a better sense of developers' work with machine learning, we asked for levels of familiarity with and use of ten ML algorithms regularly implemented in frameworks and frequently applied in data science literature. This question was optional, and far fewer developers (296) answered this question than most other questions on the survey. The results (especially the less-well-known algorithms, like random forests) are therefore less accurate than other results in this report.

A few numbers are worth contextualizing further. First, the most commonly used machine learning algorithms are linear regression and decision trees. But linear regression is a sufficiently broad concept that anyone who has ever drawn a line over a scatterplot may be loosely considered to have "used linear regression," whether or not the results were used to train a machine. Again, any programmer who has thought about control flow at all is, in some sense, "using decision trees;" but probably far fewer have used decision trees for any predictive modeling. It is precisely the conceptual overlap that makes our results difficult to interpret towards understanding developers' familiarity with machine learning techniques that underscores the affinity between programing and data science: for instance, even without explicit modeling, developers (especially while debugging) do in fact implicitly imagine rough probabilities that control will pass to certain nodes in the flow graph, given certain inputs.

Second, over half of respondents are familiar with neural nets but have not used them—by far the largest percentage of all algorithms mentioned. This suggests, perhaps, that current frameworks implementing neural nets are not easy enough to use, or that the gap between the simple concept of a neural net and applications of that concept is not sufficiently catalyzed, or that demand for very broadly domain-indifferent machine learning solutions is low compared with the supply of developers familiar with neural nets. New frameworks that simplify some of the technical issues that impose more mathematical cognitive load (TensorFlow, for example, decreases users' required comfort with linear algebra operations, while still allowing developers to build neural nets on many-dimensional data) are addressing the"catalysis" issue, but many of these frameworks are relatively new. (For more on neural nets see *What are Neural Nets?* on page 12.)

06 **HOW FAMILIAR ARE YOU WITH THESE MACHINE LEARNING ALGORITHMS?**

	USE REGULARLY	USE OCCASIONALLY	FAMILIAR BUT HAVENT USED IT	NEVER HEARD OF IT
NAÏVE BAYES CLASSIFIER	13%	16%	29%	43%
K-MEANS CLUSTERING	11%	18%	24%	47%
SUPPORT VECTOR MACHINE	5%	12%	30%	53%
APIORI ALGORITHM	3%	10%	26%	62%
LINEAR REGRESSION	16%	25%	33%	27%
LOGISTIC REGRESSION	10%	20%	30%	40%
NEURAL NETWORKS	10%	18%	52%	20%
RANDOM FORESTS	4%	16%	33%	47%
DECISION TREES	15%	31%	36%	18%
NEAREST NEIGHBORS	12%	21%	38%	30%

Data Diagnosis:

IS YOUR DATA HEALTHY?

BY DATAWATCH

SYMPTOMS YOUR DATA ISN'T WORKING FOR YOU

REKEYING IS YOUR NIGHTMARE Your data cannot easily be combined into one clean, ready-to-be-analyzed file without hours of manually rekeying data that cannot be easily extracted from sources like web pages and PDFs.

MISSING IN ACTION Your organization falls into the average metric of making decisions based on only 12% of all your data because most of it is inaccessible or 'too messy' to deal with.

DROWNING IN DATA The amount of data you have to analyze is growing so rapidly that it is difficult to keep up with. Experts predict that the amount of available data will increase by 800% in the next five years, and most of this data will be unstructured or multi-structured.

MONEY DOWN THE DRAIN Roughly 45% of operating expenses are wasted due to data quality issues. Studies estimate that organizations without data prep are wasting at least $22,000 per analyst, per year.

YOU ARE EXPOSED You don't have a system of data governance, leaving your organization unprotected from security breaches that can expose sensitive information.

NECESSARY TREATMENT FOR HEALTHIER DATA

EFFICIENCY Prepare data in minutes, not hours.

CONSOLIDATION Ability to combine, extract, and transform various file formats including TXT, HTML, XLSX, CSV, XML, JSON, Database files, unstructured and multi-structured web pages and PDFs.

ACCURACY & RELEVANCY Platform that allows analysts to analyze pertinent data, rather than data that is outdated and irrelevant.

DATA GOVERNANCE Control who has access to data to ensure privacy and precision.

AUTOMATION Save time on repeatable processes for extracting, compiling, and reporting on data.

RECOVERY FROM USING SELF-SERVICE DATA PREP TOOL

SAVED MONEY Increase revenue by focusing efforts on accurate reporting versus wasting resources on data prep.

SAVED TIME Survey results of a data preparation tool report that 78% of analysts who introduced data preparation solutions increased the amount of time they have to work with their data. 24% of these analysts gained back at least 50% of their time.

INCREASED VALUE TO YOUR ORG. Survey of Tableau users reports that 55% analysts say self-service data prep has made them 50-100% more valuable to their organizations.

INCREASED ACCESS TO 'DARK DATA' Data preparation tools allow analysts to access an increased number of previously inaccessible important pieces of data that add value and credibility to their reports.

INCREASED COMPETITIVE ADVANTAGE Organizations that invest in data prep position themselves to leverage data in a way that allows them to make better business decisions in a fraction of the time. The insight provided by data prep tools allows companies to adjust and adapt to changes in their respective industries while providing them with advantages over their toughest competitors.

OVERVIEW OF THE

Apache Spark Ecosystem

BY **FRANK EVANS**
DATA SCIENTIST AT **EXAPTIVE**

QUICK VIEW

01 Apache Spark is what finally delivers on the promise of Big Data analytics.

02 The key to understanding how to use Spark is to understand the basics of its data representation and its execution engine.

03 Spark's execution engine abstractions work best when they supplement existing powerful tools rather than trying to reinvent the wheel.

As the size of data continues to grow, the tools necessary to effectively work with it become exponentially more important. If software is eating the world, then Apache Spark has the world's biggest appetite. The primary value of Spark is its ability to control an expandable cluster of machines, and make them available to the user as though they were a single machine ecosystem. The objective of this article is to help make the under-the-hood elements of Spark less of a mystery, and to transfer existing programming knowledge and methods into the power of the Spark engine.

At the core of Spark functionality are two main components: the data storage/interaction format and the execution engine.

- Data storage and interaction is abstracted into a format Spark calls the Resilient Distributed Dataset (RDD). An RDD from the user's perspective is very similar to a single array. Each element of the array preserves its order and can be made up of virtually any object. This means an array can store scalar data (numbers, booleans, strings, etc.) as its values. This additionally means that an array can store as its values more complex data, such as tuples, sub-arrays, and even objects that represent relational data rows. Under the hood, this data format seamlessly handles partitioning data across all of the nodes in a cluster and holds them in the memory pool of the cluster as a single unit. The messy details of working with the data across multiple machines, as well as the scaling benefits, are completely abstracted away from the developer.
- Equally impressive is the execution engine. Spark includes built-in functionality for a number of common data analysis, data wrangling, and machine learning operations. It additionally includes a very clear API that allows a developer to build in their own functionality that Spark will parallelize across the entire cluster. Because of the abstraction level to the developer and the data format of the RDD, custom functions and algorithms can be easily written in such a way that is structurally similar to looping through each item of the data to do some operation, but then utilizes the Spark engine to seamlessly parallelize those operations. Then, the engine recombines the results into a single output data set, ensuring that the order of the data remains intact.

INTERACTIONS

The key to understanding how to think of a problem in a Spark mindset and how to solve a problem with the Spark ecosystem is understanding where code and data are actually processed. There are three elements in the Spark ecosystem that coordinate data processing tasks: the *driver*, the *master*, and one or more *workers*. In a normal workflow, the driver is the developer's local program that directs and coordinates the action of the entire cluster. It sends instructions to the master node on what to do. The master node will then control the workers to parallelize the data processing tasks sent from the driver. Then the driver will receive back any results it specifically asked for in its instructions.

LAZY EVALUATION

To preserve resources until absolutely necessary, the cluster nodes will not actually process instructions until requested by the driver to provide an answer that would necessitate doing that specific computation. This means operations that transform data (running a function over each element of a data set, aggregation, or anything that requires processing on the data itself) are not processed until the driver asks for a result. Once the driver asks for an answer, then all operations in the pipeline leading to that result are commenced.

PROGRAMMING LANGUAGES

- ***Scala:*** This is the core of the Spark ecosystem. Scala is a functional variant on the Java programming language that executes and compiles in the same manner as Java itself. Whichever language you use to control and develop a data processing pipeline, the underlying processing is Scala code. This additionally means that new features and capabilities in Spark are most commonly released in Scala first. However, as other languages such as Python and R gain popularity, the waiting time becomes less and less.
- ***Python & R:*** These bindings utilize both the syntax and object types of their respective domains. Since most of the coding you do will either be into functions that will be parallelized or are simply instruction sets for controlling cluster functionality, the vast majority of things you can do in Spark are available through these gateways.
- ***SQL:*** This is the newest member of the Spark team, and provides a data structure abstraction analogous to a dataframe in R or Python Pandas.

PROBLEM DOMAINS

The infrastructure and abstractions of Spark are available for any functionality that you, the developer, can write. However, there are built in Spark libraries that already have highly optimized versions of well known algorithms and machine learning pipelines that you can use right out of the box.

MACHINE LEARNING

The built-in machine learning library in Spark is broken into two parts: MLlib and KeystoneML.

- ***MLlib:*** This is the principal library for machine learning tasks. It includes both algorithms and specialized data structures. Machine learning algorithms for clustering, regression, classification, and collaborative filtering are available. Data structures such as sparse and dense matrices and vectors, as well as supervised learning structures that act like vectors but denote the features of the data set from its labels, are also available. This makes feeding data into a machine learning algorithm incredibly straightforward and does not require writing a bunch of code to denote how the algorithm should organize the data inside itself.
- ***KeystoneML:*** Like the oil pipeline it takes its name from, KeystoneML is built to help construct machine learning pipelines. The pipelines help prepare the data for the model, build and iteratively test the model, and tune the parameters of the model to squeeze out the best performance and capability.

TEXT PROCESSING

Though not as fully realized as the more numerical machine learning algorithms, there are many highly capable natural language processing (NLP) tools. For models like Latent Dirichlet Allocation topic modeling, Naive Bayes modeling, or Term Frequency-Inverse Document Frequency feature engineering, there are built-in features in MLlib. For other common NLP processes that are highly parallelizable—like stopword removal, stemming/lemmatization, or term frequency filtering—the Spark engine can efficiently run the same functions that would accomplish these tasks on a single document over millions of elements in parallel with no additional code.

GRAPH ANALYTICS

Currently only interactive from the Scala language, Spark includes a library called GraphX. This is a first-class network graph analytics engine and data object store that can run many of the standard graph analytics functions. This includes clustering, classification, traversal, searching, and path finding.

REAL-TIME STREAM PROCESSING

Just as many of the operations described so far have been batch oriented, based on static data, Spark's fast use of pooled server memory means that new data can be added to a batch at near-real-time speeds. This technique is called micro-batching, and it's the method that Spark uses for real-time data stream processing. The engine additionally provides end-to-end fault tolerance to guarantee your data will not get lost in the pipeline during processing, as well as an exactly-once guarantee that means a given portion of data in the stream will never be processed more than once.

TAKEAWAYS

Apache Spark is highly effective for big and small data processing tasks not because it best reinvents the wheel, but because it best amplifies the existing tools needed to perform effective analysis. Coupled with its highly scalable nature on commodity grade hardware, and incredible performance capabilities compared to other well known Big Data processing engines, Spark may finally let software finish eating the world.

FRANK D. EVANS is a Data Scientist with Exaptive. He primarily works with machine learning and feature engineering, specializing in unstructured and semi-structured data. His interests span natural language processing, network graph analysis, and building semi-supervised applications. Frank has a BS from St. Gregory's University and a Master's Specialization in Data Science from Johns Hopkins University.

DATA CAN BE

YOUR WORST ENEMY OR YOUR BEST FRIEND

PROBLEM: BUSINESSES DON'T HAVE ACCESS TO THE DATA THEY NEED

45% lack the big data needed in order to make major business decisions

83% of companies utilize 25% or less of their data

76% want more data sources to make more informed decisions

12% of enterprise data is used to make decisions

PROBLEM: LEVERAGING DATA IS TIME-CONSUMING & INEFFICIENT

90% of all potentially usable data needs data prep

35% of organizations have robust data prep processes

80% of an analysts' time is spent on data prep

93% of executives believe they're losing revenue due to their inability to leverage data

SOLUTION: DATAWATCH SELF-SERVICE DATA PREP

When you have the right information at the right time, you make better decisions.

Transform raw data into reliable, consistent data sets ready for analysis

Extract, cleanse, prepare & blend unworkable data into high-value data to solve business problems

Connect to any kind of data, including PDF, text reports, APIs, IoT & more

Automate & deliver prepared data to other users

DATAWATCH

Source: Economist Intelligent Unit, Oracle, Paxdata, Capgemini Global Big Data Survey, Domo

LEARN MORE AT
datawatch.com/dataprep

SELF-SERVICE DATA PREP:

Illuminating Dark Data for Better Business Decisions

Gartner defines dark data as "the information assets organizations collect, process and store during regular business activities, but generally fail to use for other purposes (for example, analytics, business relationships and direct monetizing)." When it comes to analytics, this dark data can provide immense value, and it's a critical requirement in ensuring an accurate and a holistic view of the business. So, why are data analysts and business users ignoring it?

Data that provides the most analytical value has traditionally been locked away in multi-structured or unstructured documents, such as text reports, web pages, PDFs, JSON, and log files. The only way for the average business user to use this information has been to manually rekey and reconcile the data, which are time-intensive and error-prone processes. Because the data is often deemed inaccessible, or not accessible in an acceptable timeframe, these information sources are too frequently "written off"—and hence, dark data is born.

The good news is that self-service technologies are taking data out of the dark ages.

The good news is that self-service technologies are taking data out of the dark ages and transforming how analysts and everyday business users can acquire, manipulate, and blend it. Data preparation solutions are enabling users to tap information from virtually any source, including dark data housed in previously inaccessible multi-structured and unstructured documents such as enterprise content management systems and web-based applications. Users can gain fast access to not only the right data, but all of the data, crucial to getting a holistic view of the business. This means more time can be spent on performing analysis that yields actionable business insights.

WRITTEN BY JON PILKINGTON
CHIEF PRODUCT OFFICER AT **DATAWATCH**

PARTNER SPOTLIGHT

Self-Service Data Prep

Access Any Data from Any Source, Quickly and Easily

by Datawatch

DATAWATCH

Datawatch products are unique in how they empower customers to solve some of the most difficult data challenges.

CATEGORY
Self-service data prep solutions

NEW RELEASE
Annually

OPEN SOURCE?
Yes/No

CASE STUDY

Derek Madison, Leader of Business Financial Support at MasterCard, was charged with identifying new ways to increase efficiency and improve MasterCard processes. At the outset, his team had to manually reconcile system interfaces using reports that resided on the company's mainframe. Each day they printed 20-30 individual, multi-page reports, then hand-keyed the relevant data, line by line, into Excel. "We're talking about a task that took 40-80 hours," said Madison. "We had to find a better way." After creating some reusable data prep models with Datawatch Monarch, the team was able to convert the mainframe files into a tabular data file ready for analysis. The time-intensive manual hand keying was eliminated, allowing the team to shift their resources to more strategic, valuable tasks.

STRENGTHS

- Built for business users not rocket scientists
- Automatically extract from reports, PDF documents, and web pages
- Combine, clean and use with your favorite tools
- Easily combine disparate data automatically using powerful join analysis

NOTABLE CUSTOMERS

- Mastercard
- Equifax
- Modell's Sporting Goods
- Dell
- IBM

BLOG datawatch.com/blog | **TWITTER** @Datawatch | **WEBSITE** www.datawatch.com

What are Neural Nets?

BY **EMMETT COIN**
CEO AT **ejTALK**

QUICK VIEW

01 Single-layer neural nets were invented in the 1950s, implemented by using analog hardware.

02 Brain research showed that vision centers used hidden layers of neurons, and these hidden layers necessitated back propagation algorithms.

03 Training a neural net is hard; using it is easy.

04 Deep neural nets train slowly and need more data.

We've all read about neural nets and that they're used in machine learning. Most of us know that they are in some way modeled after neurons and the way neurons are connected in the brain. But beyond that, what they are and how they actually work remains mysterious.

SOME BASICS

1) The fundamental concept behind the neuron is analog. This is a bit confusing since many of us remember from biology class that neurons communicate via pulses. All of those pulses are pretty much the same. Neurons don't transmit different amplitude pulses but instead send streams of pulses at different frequencies. It's like the difference between AM and FM radio, and it provides the same advantage of noise immunity (you can listen to FM during a thunderstorm). Neurons measure the temporal accumulation of pulses in order to determine whether they in turn emit a pulse. They do not count pulses.

2) Artificial neural nets have been around for a while. Frank Rosenblatt invented the perceptron algorithm in 1957, a special case neural net that contains a single layer of neurons. It is essentially a binary classifier. The initial algorithm was implemented on an early IBM digital computer, but because digital computers of the era were so slow, so big, and so expensive, much of the work on perceptrons was done on purpose-built electromechanical, and even electrochemical, implementations.

3) Also, at about the same time (1959 through the early 60s), David Hubel did research on living neural nets in the visual cortex of cats. Combining the unexpected results from the biological experiments with the apparent layering of interconnected neurons seen in microscopic examination of brain tissue, Hubel inferred that visual processing happened in layers of neurons that connected laterally at the same level and transmitted output signals to follow-on (deeper) intermediate layers of neurons.

4) A single layer neural net (the perceptron) is essentially a simple linear classifier. If your problem has two parameters (X and Y), and if each of the X and Y points represent one of two possible classes (A and B), then (if it is possible) the perceptron algorithm will find the equation of the straight line in the XY plane that best divides the group of points representing class A from the group of points representing class B.

5) Each neuron in a neural net has multiple inputs and one output. Associated with each input is a weighting factor (you can think of it as a variable resistor, or a volume control) which determines how much of the input signal is perceived by the neuron. All of the training that occurs in any neural net is only the adjustment of weighting factors for every input on every neuron. After the neural net is trained we record all of those weights as our solution.

6) Neural nets operate in two distinct phases:

- The learning phase in which the neural net is exposed to a lot of annotated training data. This is the hard part of the problem, and it requires a large, well-prepared, clean input dataset. The result is a straightforward (albeit potentially very large) algebraic equation. This requires a great deal of human and computer time.
- The operational phase in which parameters (annotated just like the training data was) are slotted into the algebraic equation created during the learning phase. The output of that equation can be tested against a threshold and results in an automated classification decision.

The perceptron was an interesting novelty in the late 1950s, and, as you might expect, the popular press got very excited and enthusiastically announced this technology with inappropriate hyperbole. The New York Times reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." But very early on researchers discovered that because the perceptron was just a linear classifier that it was unable to solve a wide range of problem types. Most notably Marvin Minsky denigrated the perceptron because it couldn't model a simple XOR gate. Perceptrons were abandoned for decades.

In fact, the k-means algorithm, which was one of the earliest machine learning algorithms (invented in the early 1950s), could use the same X, Y input data sets and could easily classify those points into an arbitrary number of classes. It had a big advantage over linear classifiers: a programmer could look at the code as well as the interim results and understand what was happening because it implemented the algorithm in a way that was concrete and logical. Perceptrons had a black box quality that seemed magical, and as we progressed to the modern-day deep learning nets, it's possible they have become mystical.

Because of the weaknesses of the single layer perceptron, the field stagnated for decades. But eventually the cross-pollination between the biological research into real neural nets and the computational research into simulated feedback and highly connected logical networks sparked the invention of multilayer perceptron, which we know today as a neural network. Most of the work in the 1990s experimented with three layer networks:

1) The input layer, which accepted the input parameters (essentially a perceptron).

2) The hidden layer, which accepted the outputs of the input layer (transformations of the original input parameters).

3) The output layer, which accepted the outputs of the hidden layer and results in the output classification.

It seemed likely to the researchers that an intermediate layer of neurons might provide a computational advantage. After all, nature used multiple layers of neurons, and the experiments of Hubel suggested that the deeper layers computed a generalized version of the original input. But training a neural net with a hidden layer introduced some serious challenges. From the beginning, the core method for training a single layer neural net (perceptron) was a technique called gradient descent:

1) For each data point: apply parameters from one of the data points.

2) For each input:

   A. Adjust the input weight by a tiny amount.

   B. Test the actual output against the expected output.

      I. If it is closer, keep the new weight value.

      II. If it is farther, make the change in the other direction and keep that new weight value.

Training a neural net involves doing this repeatedly over all of the training data until the average error between the expected output and the actual output plateaus. Of course, nothing is that simple, but these ancillary problems could be tackled with well-known techniques (e.g. stochastic weight value changes). The new problem specific to these new hidden layer networks was the less direct connection between input and output. The causality between input values was obscured or at least blurred by the hidden layer. How could you adjust the weights on the inputs of the hidden layer neurons in a sensible and efficient way so that it would converge on a solution? We could measure the error between the expected and actual output, but how could we propagate the desired correction back to the hidden layer input weights and ultimately to the raw input layer weights? This became a major pillar of research upon which multilayer neural, and in the extreme, deep learning networks evolved.

BACKPROPAGATION

This problem was worked by a number of research groups and was invented independently a number of times. The core concept of backpropagation is to more accurately predict the amount of change needed for each weight by knowing the correlation between the local rate of change of the weight and the rate of change of the error it induces. This implies that we are operating on the derivatives of the weight and error changes. Because both of those must be differentiable, we can no longer use simple thresholds for the neuron activation. This is why the activation function becomes important, and many, if not most, neural networks use the logistic function because, in the extreme, it gives a binary output (0 to 1), but it is forgiving (differentiable) as it transitions between those values. It's as if the neuron outputs are graded on a curve, not just pass-fail. In essence the output layer is able to indicate to the hidden layer how much influence its recent weight changes are having on the output. Without backpropagation, multilayer neural nets would be at best computationally impractical and at worst impossible.

As you might guess, neural nets with hidden layers take longer to train. Not only do we have more neurons and thus more weights to adjust, they must also do more computation for each of those weights. In addition, because of the blurring of the direct causality relationship while working through the hidden layer, the system needs to take smaller steps to ensure that the gradient descent stays on track. And that means more passes over the training set. This problem only gets worse as we progressed to deep learning systems that have many hidden layers (I have seen up to six). To get back to a biological reference, your cerebral cortex has six distinct layers.

So, if we continue to take our AI hints from biology, then perhaps our deep learning models are on the right track!

EMMETT COIN is the founder of ejTalk where he researches and creates engines for human-computer conversation systems to manage real-time, intelligent, and natural conversations with humans. He has worked in speech technology since the 1960s at MIT. He is a board member for AVIOS and the Advanced Dialog Group. His research focuses on the tension between a machine learning approach and a declarative rule-based approach. His goal is to model the innate mechanisms of generic conversational skill and apply this to specific domains.

THE DO'S AND DON'TS OF

Using Regexes With Big Data

BY **ELIZABETH BENNETT**
SENIOR SOFTWARE ENGINEER AT **LOGGLY**

QUICK VIEW

01 Regular expressions are extremely powerful, as well as elegant, simple, and universal. Even in today's world of Big Data, regular expressions have a perennial place in any software engineer's toolkit.

02 Formal regular expressions are not the same thing as the regexes we know now. The two terms are often conflated, but over the years, regexes have evolved to be more powerful than formal regular expression, starting with Perl's regex engine.

03 If you are using regexes to parse huge amounts of data, a few small adjustments in the regex definition can make the difference between needing 20 machines to do the processing and needing 1 machine.

Regular expressions have been around for decades. They predate almost all of the tools and technologies we talk about in today's world of Big Data, scalability, slick UIs, and machine learning. Commonly viewed as cryptic, frustrating, and difficult to learn, many developers scoff at the thought of using regular expressions for anything besides validating an e-mail address. Regular expressions are extremely powerful, however, as well as elegant, simple, and universal. Even in today's world of Big Data, regular expressions have a perennial place in any software engineer's toolkit. According to formal language theory, regular expressions are as fundamental to computer science as any programming language or machine-readable data format. They are the mechanism by which unstructured data becomes structured. They turn chaos into order.

When faced with a text processing challenge, few tools are better suited to the task than regular expressions. At a certain scale however, there are performance caveats to be aware of. To explore those caveats, let's first dive into a bit of computer science theory.

What exactly is a regular expression? A regular expression formally is the textual encoding of a finite state machine. Because of this, their expressive power is limited, but they process input extremely efficiently. They cannot, for example, determine whether or not an input is a palindrome, whereas the regular expression's more powerful cousin, the context-free grammar, can determine whether a string is a palindrome. Regular expressions are strictly less powerful than context-free grammars, which in turn are less powerful than Turing complete languages, like C or Java. Thus, anything you could do with a regular expression, you could technically also do with a standard programming language.

Strictly speaking, regular expressions are not the same thing as the regex we know now. The two terms are often conflated, but over the years regex has evolved to be more powerful than formal regular expression, starting with Perl's Regex engine. Features such as capture group back references are not technically possible with actual regular expressions. True regular expressions can be processed quickly because they represent finite state machines and the engines that support these regular expressions use the extremely efficient Thompson NFA algorithm (e.g. SED, AWK, and GREP). When regexes stopped representing formal regular expressions, a new algorithm was devised: the recursive backtracking algorithm. The advent of this new algorithm introduced a slew of regex performance gotchas, which are now almost a defining characteristic of regexes.

With recursive backtracking based regex engines, it is possible to craft regular expressions that match in exponential time with respect to the length of the input, whereas the Thompson NFA algorithm will always match

in linear time. As the name would imply, the slower performance of the recursive backtracking algorithm is caused by the backtracking involved in processing input. This backtracking has serious consequences when working with regexes at a high scale because an inefficient regex can take orders of magnitude longer to match than an efficient regex. The standard regex engines in most modern languages, such as Java, Python, Perl, PHP, and JavaScript, use this recursive backtracking algorithm, so almost any modern solution involving regexes will be vulnerable to poorly performing regexes. Fortunately, though, in almost all cases, an inefficient regex can be optimized to be an efficient regex, potentially resulting in enormous savings in terms of CPU cycles.

To illustrate this, let's consider an example use case: parsing log data. Due to their predictable format, log lines can easily be matched against regexes to extract useful pieces of information. Let's say we use the following example regex to parse log lines:

```
.* (.*)\[(.*)\]:.*
```

The example log lines might look something like this:

```
2014-08-26 app[web.1]: 50.0.134.125 - - [26/
Aug/2014 00:27:41] "GET / HTTP/1.1" 200 14 0.0005
```

The information we're interested in gathering is `app` and `web.1`, which are extracted via the first and second capture groups. Although the above regex is simple, readable, and sufficient for extracting the needed information from the input text, a far more efficient regex would look like this:

```
[12]\d{3}-[01]\d-[0-3]\d ([^ \[]*)\[([^\]]*)\]:.*
```

Both regular expressions extract the same data from the same input text. The second regular expression, however, is far more specific about the format of the data it expects to match. It first encodes the format of the timestamp of the expected log lines, which will almost instantly rule out log lines that do not match. It also uses negated character classes rather than `.*`'s which eliminates almost all backtracking. The inefficient regular expression, because of all of its `.*`'s, backtracks excessively, which is what causes it to match so much more slowly. Since it begins with a `.*`, it will consume as much text as it can, zooming down the input until it reaches the end. Then it will search backwards until it reaches the next character it's looking for, which in this case is a space character. Once it encounters the first space from the end, it again consumes all characters until the end of the input and backtracks again in search of the next character, which is an open square bracket. It will reach the end and it won't find an open bracket, so it goes back to looking for spaces, hoping that if it had consumed less text in the first place, maybe it could still match the input. This process continues until either the input matches or there are no more possibilities to try. This is the recursive backtracking process that causes the exponential runtime characteristics. The efficient regular expression on the other hand, does none of this backtracking and will basically consume characters from left to right.

A simple benchmark where each regex is fed the sample input 10,000 times shows that the inefficient regex takes about 20 times longer to match than the efficient regex. This is pretty astounding since both regexes extract the same information from the same input. If you are using regexes to parse huge amounts of data, a few small adjustments in the regex definition can make the difference between needing 20 machines to do the processing, and needing 1 machine. This, obviously, translates to serious cost and latency savings.

If you are using regexes to parse huge amounts of data, a few small adjustments in the regex definition can make the difference between needing 20 machines to do the processing, and needing 1 machine

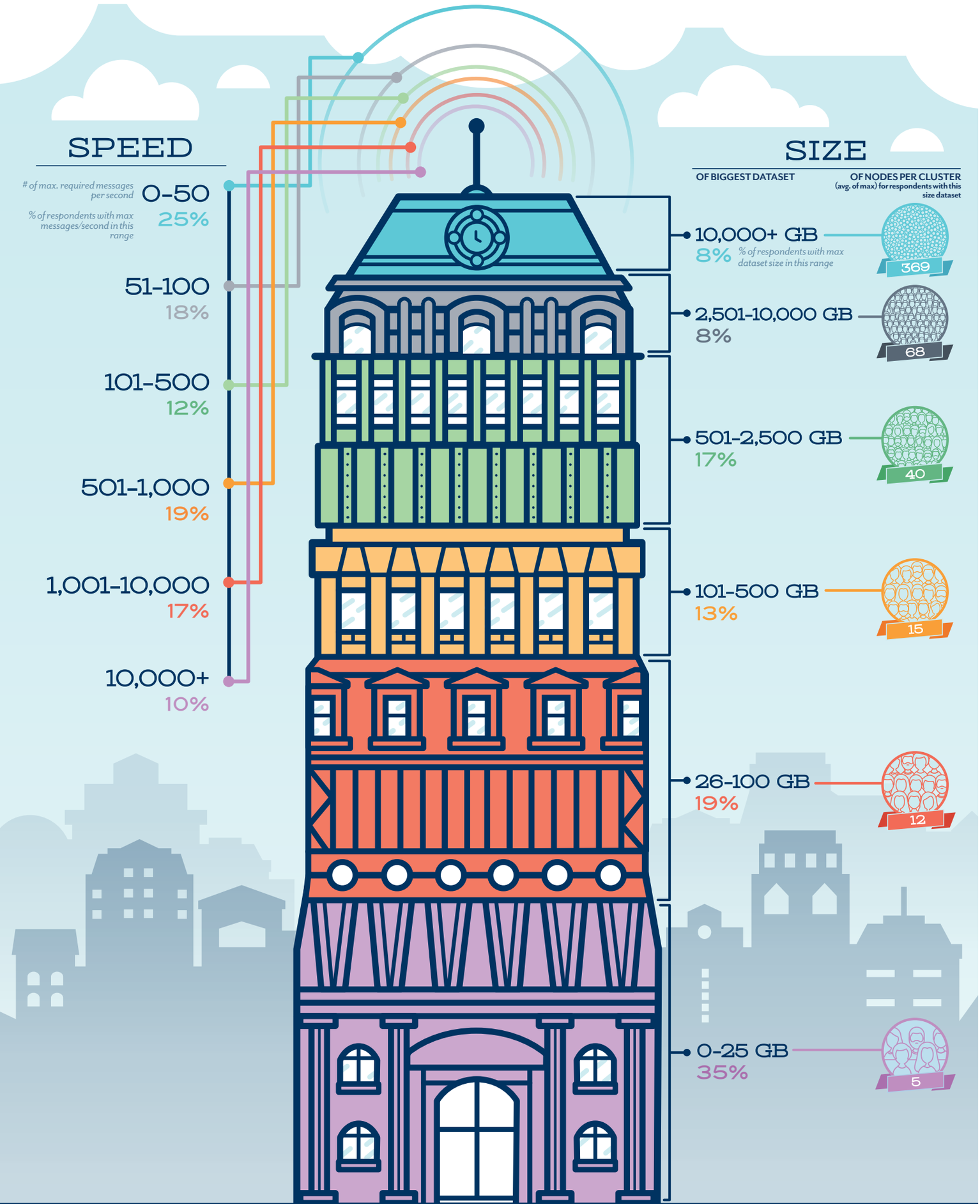
Although regular expressions are not technically as powerful as, say, the Java code you may be writing, regular expressions are simple and effective at doing what they do. A regex engine is a black box that uses patterns that are understood by almost every developer. Every modern language contains a regex engine and most of those regex engines have been highly optimized to crunch through data as quickly as possible. It's almost unfortunate that the standard regex engines in today's languages have such serious performance caveats. Perhaps given the sorts of use cases that regexes have in the world of Big Data, the time is ripe for the emergence of standard regex engines that are not powered by the recursive backtracking algorithm but by the Thompson NFA algorithm. Such an engine would be less powerful in terms of the types of input it could match, but would make up for it with its vastly superior and consistent performance. Until then, we developers will continue to use regexes to great effect, but we will use them with a discerning eye, ever vigilant to performance gotchas and always performance testing our regexes to ensure we are getting the optimum performance.

ELIZABETH BENNETT is a Senior Software Engineer at Loggly. A Denver native, she moved to the Bay Area shortly after graduating from Oberlin College with degrees in Computer Science and Bassoon Performance (en.wikipedia.org/wiki/Bassoon, for those who have no clue what a bassoon is). While most of her days are spent writing Java for Loggly's backend services, she does manage to spend a fair amount of time knitting and video gaming. Also, it is in fact true that Jane Austen named her character after Elizabeth.

FROM DZONE.COM

BUILDING *Big Data*

It takes a lot of resources to build a system for big and fast data. How much? To paint an empirical picture of how developers handle real-world data volume and velocity, we asked: (1) how big (in GB) is your biggest dataset? (2) how many messages (max) do you have to process per second? and (3) how many nodes (max) are in your data processing clusters (cross-tabbed with biggest dataset size)? Results of 437 responses are summarized below.



Got Data?

Get Qubole. The Most Powerful Platform for Big Data Teams

Qubole

Brands that depend on Qubole

ORACLE | Datalogix lyft Flipboard Pinterest UNDER ARMOUR

What is Qubole?

Data Science Optimization

Qubole's auto-scaling and automated cluster lifecycle management eliminate barriers to data science.

Spark Notebooks in One Click

Combine code execution, rich text, mathematics, plots, and rich media for self-service data exploration.

Become Data Driven

Give your team access to Machine Learning algorithms, ETL with Hive, and real-time processing using Spark.

Try Qubole Today

Optimizing Big Data Cluster Performance and Cost in the Cloud

Qubole, a Big-Data-as-a-service platform designed to enhance speed, scalability, and accessibility around Big Data projects, recently undertook a benchmarking analysis of their auto-scaling feature. They compared an auto-scaling Spark cluster to static Spark clusters, simulating a real customer's workload. The analysis showed that auto-scaling clusters return queries faster and use fewer node hours than equivalently sized static clusters. In a follow-up, Qubole found that auto-scaling was saving one of our customers $300,000 in compute hours per year.

What we all really want from our Big Data clusters is for them to always fit, no matter how workloads change. Underutilized clusters waste money, overburdened clusters delay query results, and the cluster management needed to manually ensure optimal cluster size wastes time and money.

Workloads are elastic. To survive and thrive, clusters must embrace that elasticity. The ideal Big Data cluster should be like Spandex, not a suit that has to be taken to the tailor.

Auto-scaling saves money and time by adding or subtracting nodes automatically to suit the current workload, letting cluster administrators sit back and not worry about mis-sizing a cluster. The admin simply sets the minimum and maximum number of nodes that can be used, and the auto-scaling algorithm does the rest. Today, companies like Autodesk, Pinterest, and Oracle's Datalogix rely on Qubole's auto-scaling to deliver savings and rapid query results.

Workloads are elastic. To survive and thrive, clusters must embrace that elasticity.

No one would be surprised if Spandex was the name of the next Hadoop project, but in fact it's just a fabric that has a lot to teach us about how to construct a flexible, efficient analytics cluster.

Visit Qubole's blog here.

WRITTEN BY DHARMESH DESAI
TECHNOLOGY EVANGELIST AT **QUBOLE**

PARTNER SPOTLIGHT

Qubole Data Service by Qubole

The Qubole Data Service is the leading Big Data-as-a-service platform for the cloud

CATEGORY
Big Data

NEW RELEASE
Continuously updated SaaS

OPEN SOURCE?
Yes – Hybrid

OVERVIEW

The Qubole Data Service (QDS) provides data teams the control, automation and orchestration to deploy and run any kind of big data workload, using any analytic engine for an unlimited number of users on their choice of Amazon AWS, Microsoft Azure or Google Cloud Platform. It supports a full range of data processing engines: Hadoop, Spark, Presto, Hbase, Hive and more. Analysts and Data Scientists use the Workbench features to run ad hoc queries and machine learning while data engineers use QDS for ETL and other batch workloads. QDS is the only cloud-native big data platform and provides cluster management through a control panel or API to make administrative tasks easy, performance optimization via auto-scaling and cost control through policy based spot instance support.

QDS is used by firms such Pinterest, Oracle and Under Armour to process over 300 petabytes of data monthly.

STRENGTHS

- Cloud-native platform runs on Amazon AWS, Microsoft Azure and Google Cloud Platform
- Supports most big data processing engines: Hadoop, Spark, Presto, Hive, Pig, Hbase and more
- Easy to deploy SaaS requires no hardware or software install, run workloads immediately
- Optimizes cost and performance with auto-scaling and spot-instance features
- Automates cluster management through policy based configuration management

NOTABLE CUSTOMERS

- Datalogix
- Pinterest
- Flipboard
- Autodesk
- Under Armour
- Lyft

BLOG qubole.com/big-data-blog | **TWITTER** @qubole | **WEBSITE** www.qubole.com

MapReduce Design Patterns

BY **SHITAL KATKAR**
SOFTWARE DEVELOPER AT **PI TECHNIQUES**

QUICK VIEW

01 MapReduce basically uses a "divide and conquer" strategy. It primarily involves the two functions Map and Reduce.

02 It is not necessarily true that every Map job is followed by a Reduce job. It depends upon the situation.

03 To solve any problem in MapReduce, we need to think in terms of Map and Reduce.

04 This article presents some basic patterns, but you can adjust these patterns as needed for the problem you are trying to solve.

This article discusses four primary MapReduce design patterns:

1. **Input-Map-Reduce-Output**
2. **Input-Map-Output**
3. **Input-Multiple Maps-Reduce-Output**
4. **Input-Map-Combiner-Reduce-Output**

Following are some real-world scenarios, to help you understand when to use which design pattern.

INPUT-MAP-REDUCE-OUTPUT

If we want to perform an aggregation operation, this pattern is used:

Scenario	Counting the total salary of employees based on gender.
Map (Key, Value)	Key : Gender Value : Their Salary
Reduce	Group by Key (Gender), then take the sum of the value (salary) for each group.

EMPLOYEE NAME	GENDER	SALARY IN K
Krishna	Male	80
Namrata	Female	75
Ritesh	Male	100
Ram	Male	40
Vijay	Male	250
Jitu	Male	55
Ria	Female	90
Poonam	Female	300
Radha	Female	40

To count the total salary by gender, we need to make the key Gender and the value Salary. The output for the Map function is:

```
(Male, 80), (Female, 75), (Male, 100), (Male, 40), (Male, 250),
(Male, 55), (Female, 90), (Female, 300), (Female, 40)
```

Intermediate splitting gives the input for the Reduce function:

```
(Male <80, 100, 40, 250, 55>), (Female<75, 90, 300, 40>)
```

And the Reduce function output is:

```
(Male, 525), (Female, 505)
```

INPUT-MAP-OUTPUT

The Reduce function is mostly used for aggregation and calculation. However, if we only want to change the format of the data, then the Input-Map-Output pattern is used:

Scenario	The data source has inconsistent entries for Gender, such as "Female", "F", "f", or 0. We want to make the column consistent, i.e., for male, "M" should be used, and for female, "F" should be used.
Map (Key, Value)	Key : Employee ID Value : Gender -> If Gender is Female/female/ F/ f/ 0, then return F; else if Gender is Male/male/M/m/1, then return M.

In the Input-Multiple Maps-Reduce-Output design pattern, our input is taken from two files, each of which has a

different schema. (Note that if two or more files have the same schema, then there is no need for two mappers. We can simply write the same logic in one mapper class and provide multiple input files.)

Scenario 1	We have to find the total salary by gender. But we have 2 files with different schemas. **Input File 1** Gender is given as a prefix to the name. *E.g. Ms. Shital Katkar* *Mr. Krishna Katkar* **Input File 2** There is a separate, dedicated column for gender. However, the format is inconsistent. *E.g. Female/Male, 0/1, F/M*
Map (Key, Value)	**Map 1 (for input 1)** We need to write program to split each prefix from its name and determine the gender according to the prefix. Then we prepare the key-value pair (Gender, Salary). **Map 2 (for input 2)** Here, the program will be straightforward. Resolve the mixed formats (as seen earlier) and make a key-value pair (Gender, Salary).
Reduce	Group by Gender and take the total salary for each group.

This pattern is also used in **Reduce-Side Join**:

Scenario 2	There are two large files, 'Books and 'Rating'. **Book ID** | **Book Name** 101 | Classical Mythology 102 | The Kitchen God's Wife 103 | Haveli **Book ID** | **Rating** | **User ID** 101 | 8 | X134 102 | 7 | D454 101 | 7 | C455 103 | 6 | B455 We want to display each book and its average rating.
Map (Key, Value)	In this case, we have to join two tables, Books and Rating, on the Book ID column, so we open the two files in two different Map functions. **Map 1 (for the Books table)** Since we want to join on Book ID, it should be sent as the key in both maps. Key-value pair: (Book ID, Book Name) **Map 2 (for the Rating table)** Key-value pair: (Book ID, Rating)
Splitting Output	(101< Classical Mythology, 8, 7, 9, 6,...,...,...,...,...>) (102<The Kitchen God's Wife,7,8,5,6,.....>) (103<Haveli,6,7,5,6,.....>)
Reduce	Group By Book ID, take the average of Rating, and return (Book Name, Average Rating). (Classical Mythology, 7.5) (The Kitchen God's Wife, 6.5) (Haveli, 6)

INPUT-MAP-COMBINER-REDUCE-OUTPUT

Apache Spark is highly effective for big and small data processing tasks not because it best reinvents the wheel, but because it best amplifies the existing tools needed to perform effective analysis. Coupled with its highly scalable nature on commodity grade hardware, and incredible performance capabilities compared to other well known Big Data processing engines, Spark may finally let software finish eating the world.

A Combiner, also known as a semi-reducer, is an optional class that operates by accepting the inputs from the Map class and then passing the output key-value pairs to the Reducer class. The purpose of the Combiner function is to reduce the workload of Reducer.

In a MapReduce program, 20% of the work is done in the Map stage, which is also known as the data preparation stage. This stage does work in parallel.

80% of the work is done in the Reduce stage, which is known as the calculation stage. This work is not done in parallel, so it is slower than the Map phase. To reduce computation time, some work of the Reduce phase can be done in a Combiner phase.

SCENARIO

There are five departments, and we have to calculate the total salary by department, then by gender. However, there are additional rules for calculating those totals. After calculating the total for each department by gender:

If the total department salary is greater than 200K → add 25K to the total.

If the total department salary is greater than 100K → add 10K to the total.

Input Files (for each dept.)	**Map** (Parallel) (Key = Gender, Value = Salary)	**Combiner** (Parallel)	**Reducer** (Not parallel)	**Output**
Dept. 1	Male <10,20,25,45,15,45,25,20> Female <10,30,20,25,35>	Male <250,25> Female <120,10>	Male < 250,25,155, 10,90,90,30> Female <120,10,175,10,135, 10,110,10,130,10>	Male <650> Female <720>
Dept. 2	Male<15,30,40,25,45> Female <20,35,25,35,40>	Male <155,10> Female <175,10>		
Dept. 3	Male<10,20,20,40> Female <10,30,25,70>	Male <90,00> Female <135,10>		
Dept. 4	Male<45,25,20> Female <30,20,25,35>	Male <90,00> Female <110,10>		
Dept. 5	Male<10,20> Female <10,30,20,25,35>	Male <30,00> Female <130,10>		

SHITAL KATKAR is a graduate student at Mumbai University, Mastering in Computer Applications with a major focus on BI techniques such as data mining, data analytics, and web applications. She currently works as a software developer in Mumbai, India. She is a passionate evangelist of technology, and enjoys sharpening her skills to move into Big Data analytics. Her main areas of interest include writing programs with MapReduce and Python to manage and analyze data.

Building High-Performance

BIG DATA AND ANALYTICS SYSTEMS

QUICK VIEW

01 Understand the building blocks of big data systems

02 Key performance considerations for each building block

03 Understand how security requirements impact the performance of the system

BY **ROHIT DHALL**
ENTERPRISE ARCHITECT AT **HCL TECHNOLOGIES**

Big Data and Analytics systems are fast emerging as one of the most critical system in an organization's IT environment. But with such a huge amount of data, there come many performance challenges. If Big Data systems cannot be used to make or forecast critical business decisions, or provide insights into business values hidden under huge amounts of data at the right time, then these systems lose their relevance. This article talks about some of the critical performance considerations in a technology-agnostic way. These should be read as generic guidelines, which can be used by any Big Data professional to ensure that the final system meets all performance requirements.

BUILDING BLOCKS OF A BIG DATA SYSTEM

A Big Data system is comprised of a number of functional blocks that provide the system the capability for acquiring data from diverse sources, pre-processing (e.g. cleansing and validating) this data, storing the data, processing and analyzing this stored data, and finally presenting and visualizing the summarized and aggregated results.

The rest of this article describes various performance considerations for each of the components shown in Figure 1.

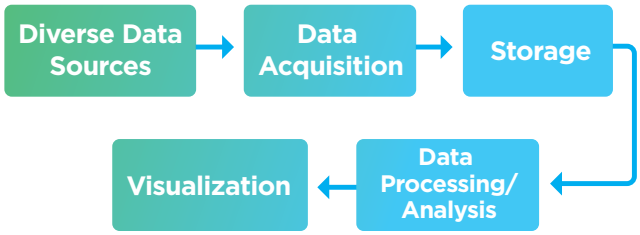


Fig. 1: Building blocks of a Big Data System

PERFORMANCE CONSIDERATIONS FOR DATA ACQUISITION

Data acquisition is the step where data from diverse sources enters the Big Data system. The performance of this component directly impacts how much data a Big Data system can receive at any given point of time.

Some of the logical steps involved in the data acquisition process are shown in the figure below:

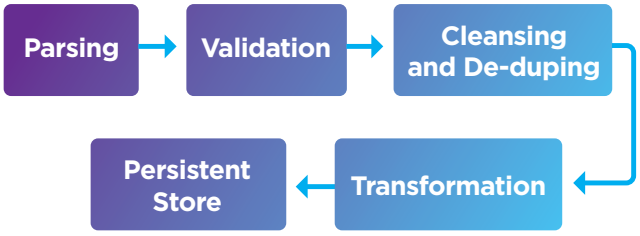


Fig. 2: Data Acquisition

The following list includes some of the performance considerations, which should be followed to ensure a well performing data acquisition component.

- Data transfer from diverse sources should be asynchronous. Some of the ways to achieve this are to either use file-feed transfers at regular time intervals or by using Message-Oriented-Middleware (MoM). This will allow data from multiple sources to be pumped in at a much faster rate than what a Big Data system can process at a given time.
- If data is being parsed from a feed file, make sure to use appropriate parsers. For example, if reading from an XML file, there are different parsers like JDOM, SAX, DOM, and so on. Similarly for CSV, JSON, and other such formats, multiple parsers and APIs are available.
- Always prefer to see built-in or out-of-the-box validation solutions. Most parsing/validation workflows generally run in a server environment (ESB/AppServer). These have standard validators available for almost all scenarios. Under most circumstances, these will generally perform much faster than any custom validator you may develop.
- Identify and filter out invalid data as early as possible, so that all the processing after validation will work only on legitimate sets of data.
- Transformation is generally the most complex and the most time- and resource-consuming step of data acquisition, so make sure to achieve as much parallelization in this step as possible.

PERFORMANCE CONSIDERATIONS FOR STORAGE

In this section, some of the important performance guidelines for storing data will be discussed. Both storage options—logical data storage (and model) and physical storage—will be discussed.

- Always consider the level of normalization/de-normalization you choose. The way you model your data has a direct impact on performance, as well as data redundancy, disk storage capacity, and so on.
- Different databases have different capabilities: some are good for faster reads, some are good for faster inserts, updates, and so on.
- Database configurations and properties like level of replication, level of consistency, etc., have a direct impact on the performance of the database.
- Sharding and partitioning is another very important functionality of these databases. The way sharding vis configured can have a drastic impact on the performance of the system.
- NoSQL databases come with built-in compressors, codecs, and transformers. If these can be utilized to meet some of the requirements, use them. These can perform various tasks like formatting conversions, zipping data, etc. This will not only make later processing faster, but also reduce network transfer.
- Data models of a Big Data system are generally modeled on the use cases these systems are serving. This is in stark contrast to RDMBS data modeling techniques, where the database model is designed to be a generic model, and foreign keys and table relationships are used to depict real world interactions among entities.

Identify and filter out invalid data as early as possible, so that all the processing after validation will work only on legitimate sets of data.

PERFORMANCE CONSIDERATIONS FOR DATA PROCESSING

This section talks about performance tips for data processing. Note that depending upon the requirements, the Big Data system's architecture may have some components for both real-time stream processing and batch processing. This section covers all aspects of data processing, without necessarily categorizing them to any particular processing model.

- Choose an appropriate data processing framework after a detailed evaluation of the framework and the requirements of the system (batch/real-time, in-memory or disk-based, etc.).
- Some of these frameworks divide data into smaller chunks. These smaller chunks of data are then processed independently by individual jobs.
- Always keep an eye on the size of data transfers for job processing. Data locality will give the best performance because data is always available locally for a job, but achieving a higher level of data locality means that data needs to be replicated at multiple locations.
- Many times, re-processing needs to happen on the same set of data. This could be because of an error/exception in initial processing, or a change in some business process where the business wants to see the impact on old data as well. Design your system to handle these scenarios.

- The final output of processing jobs should be stored in a format/model, which is based on the end results expected from the Big Data system. For example, if the expected end result is that a business user should see the aggregated output in weekly time-series intervals, make sure results are stored in a weekly aggregated form.
- Always monitor and measure the performance using tools provided by different frameworks. This will give you an idea of how long it is taking to finish a given job.

Choose an appropriate data processing framework after a detailed evaluation of the framework and the requirements of the system (batch/real-time, in-memory or disk-based, etc.).

PERFORMANCE CONSIDERATIONS FOR VISUALIZATION

This section will present generic guidelines that should be followed while designing a visualization layer.

- Make sure that the visualization layer displays the data from the final summarized output tables. These summarized tables could be aggregations based on time-period recommendations, based on category, or any other use-case-based summarized tables.
- Maximize the use of caching in the visualization tool. Caching can have a very positive impact on the overall performance of the visualization layer.
- Materialized views can be another important technique to improve performance.
- Most visualization tools allow configurations to increase the number of works (threads) to handle the reporting requests. If capacity is available, and the system is receiving a high number of requests, this could be one option for better performance.
- Keep the pre-computed values in the summarized tables. If some calculations need to be done at runtime, make sure those are as minimal as possible, and work on the highest level of data possible.
- Most visualization frameworks and tools use Scalable Vector Graphics (SVG). Complex layouts using SVG can have serious performance impacts.

BIG DATA SECURITY AND ITS IMPACT ON PERFORMANCE

Like any IT system, security requirements can also have a serious impact on the performance of a Big Data system. In this section, some high-level considerations for designing security of a Big Data system without having an adverse impact on the performance will be discussed.

- Ensure that the data coming from diverse sources is properly authenticated and authorized at the entry point of the Big Data system.
- Once data is properly authenticated, try to avoid any more authentication of the same data at later points of execution. To save yourself from duplicate processing, tag this authenticated data with some kind of identifier or token to mark it as authenticated, and use this info later.
- More often than not, data needs to be compressed before sending it to a Big Data system. This makes data transfer faster, but due to the need of an additional step to un-compress data, it can slow down the processing.
- Different algorithms/formats are available for this compression, and each can provide a different level of compression. These different algorithms have different CPU requirements, so choose the algorithm carefully.
- Evaluate encryption logic/algorithms before selecting one.
- It is advisable to keep encryption limited to the required fields/information that are sensitive or confidential. If possible, avoid encrypting whole sets of data.

Ensure that the data coming from diverse sources is properly authenticated and authorized at the entry point of the Big Data system.

CONCLUSION

This article presented various performance considerations, which can act as guidelines to build high-performance Big Data and analytics systems. Big Data and analytics systems can be very complex for multiple reasons. To meet the performance requirements of such a system, it is necessary that the system is designed and built from the ground up to meet these performance requirements.

ROHIT DHALL is an Enterprise Architect with the Engineering and R&D Services division of HCL Technologies. His main area of expertise is architecting, designing, and implementing high-performance, fault-tolerant, and highly available solutions for leading Telco and BFSI organizations. He writes white papers, articles, and blogs for various IT events, forums, and portals. He is also a co-author of the IBM Redbook and Redpaper on "ITCAM for WebSphere".

Diving Deeper

INTO BIG DATA

TOP BIG DATA TWITTER FEEDS

 @matei_zaharia

 @BigDataGal

 @KirkDBorne

 @medriscoll

 @spyced

 @BigData_paulz

 @kdnuggets

 @jameskobielus

 @marcusborba

 @data_nerd

BIG DATA ZONES

Big Data Zone

dzone.com/big-data

The Big Data/Analytics Zone is a prime resource and community for Big Data professionals of all types. We're on top of all the best tips and news for Hadoop, R, and data visualization technologies. Not only that, but we also give you advice from data science experts on how to understand and present that data.

Database Zone

dzone.com/database

The Database Zone is DZone's portal for following the news and trends of the database ecosystems, which include relational (SQL) and non-relational (NoSQL) solutions such as MySQL, PostgreSQL, SQL Server, NuoDB, Neo4j, MongoDB, CouchDB, Cassandra and many others.

IoT Zone

dzone.com/iot

The Internet of Things (IoT) Zone features all aspects of this multifaceted technology movement. Here you'll find information related to IoT, including Machine to Machine (M2M), real-time data, fog computing, haptics, open distributed computing, and other hot topics. The IoT Zone goes beyond home automation to include wearables, business-oriented technology, and more.

TOP BIG DATA REFCARDZ

R Essentials

dzone.com/refcardz/r-essentials-1

Introduces R, the language for data manipulation and statistical analysis.

Distributed Machine Learning with Apache Mahout

dzone.com/refcardz/distributed-machine-learning

Introduces Mahout, a library for scalable machine learning, and studies potential applications through two Mahout projects.

Practical Data Mining with Python

dzone.com/refcardz/data-mining-discovering-and

Covers the tools used in practical Data Mining for finding and describing structural patterns in data using Python.

TOP BIG DATA WEBSITES

Dataversity.net

Daily updates on the latest in Big Data news and research, including cognitive computing, business intelligence, and NoSQL.

PlanetBigData.com

Content aggregator for Big Data blogs with frequent updates.

DataScienceCentral.com

A site by practitioners for practitioners that focuses on quality information over marketing ploys.

TOP BIG DATA PAPERS

100 Open Source Big Data Architecture Papers for Data Professionals

bit.ly/2aYhmge

Optimizing Big Data Analytics on Heterogeneous Processors

bit.ly/2aYjlvQ

Challenges and Opportunities with Big Data

bit.ly/2blilGF

Executive Insights

ON BIG DATA

BY **TOM SMITH**
RESEARCH ANALYST AT **DZONE**

QUICK VIEW

01 With the growth of IoT devices, be prepared to handle more data, faster, from more sources, and in more forms.

02 Spark has supplanted Hadoop as the Big Data platform of preference because it analyzes and hands off datasets more quickly.

03 Expect Big Data, combined with machine learning, to anticipate/predict problems, and fix them, before they even occur.

To gather insights on the state of Big Data today, we spoke with 15 executives providing Big Data products and services to clients. Specifically, we spoke to:

Uri Maoz, Head of U.S. Sales and Marketing, Anodot
Dave McCrory, CTO, Basho
Carl Tsukahara, CMO, Birst
Bob Vaillancourt, Vice President, CFB Strategies
Mikko Jarva, CTO Intelligent Data, Comptel
Sham Mustafa, Co-Founder and CEO, Correlation One
Andrew Brust, Senior Director Marketing Strategy, Datameer
Tarun Thakur, CEO/Co-Founder, Datos IO
Guy Yehiav, CEO, Profitect
Hjalmar Gislason, Vice President of Data, Qlik
Guy Levy-Yurista, Head of Product, Sisense
Girish Pancha, CEO, StreamSets
Ciaran Dynes, Vice Presidents of Products, Talend
Kim Hanmark, Director, Professional Services, TARGIT
Dennis Duckworth, Director of Product Marketing, VoltDB.

KEY FINDINGS

01 The keys to working with Big Data are to be prepared for: **1) the number of sources from which data is coming; 2) the high volume of data; 3) the different forms of data; 4) the speed with which the data is coming; and, 5) the elasticity of the database, the enterprise, and the applications.** Innovative solutions are becoming available to address the volume and elasticity of data, as well as the integration of the different data types (unstructured, semi-structured, and structured) from different sources. The key is to understand up front what you want to get from the data and then to plan accordingly to ensure the data is operationalized to provide value for the end user—typically in a corporate environment.

02 The most significant changes to Big Data tools and technologies in the past year have been: **1) Spark supplanting MapReduce and Hadoop; 2) machine learning and clustered computing coming to the forefront; 3) the cloud enabling larger datasets at ever lower prices; and, 4) the new tools that make the analysis of large, disparate datasets even faster.** Spark has sucked the energy out of some of the newer frameworks while Google cloud has made machine learning and artificial learning accessible. A lot of companies are using Apache Spark as their Big Data platform because it analyzes and hands-off datasets more quickly. The cloud has provided prominent deployment options for companies not in the IT business—Big Data is an operating expense rather than a capital expense so it's easier to get funding. Cloud and Big Data go hand-in-hand, and the cloud providers are ensuring this by developing tools that make Big Data accessible to business professionals rather than just data scientists.

03 Our 15 respondents mentioned 29 different technical solutions they use on Big Data projects. The most frequently mentioned technical solutions are: **1) Open Source; 2) Apache Spark; 3) Hadoop; 4) Kafka; and, 5) Python**.

04 Our respondents provided a wide array of use cases and examples of how Big Data is being used to solve

real-world problems. The most frequently mentioned use cases involve: **1) real-time analytics; 2) IoT; and, 3) predictive analytics**. Real-time analytics are being used by e-commerce and telcos to provide more personalized services, dynamic pricing, and customer experiences. It's clear real-time data is more valuable to clients and end users, and as such the speed of ingestion and analysis is key. IoT is most prevalent in industry and utilities to track the use, production, predictive maintenance, and outages to optimize performance, productivity, and efficiency. Predictive analytics are being used for maintenance to reduce downtime in airlines, turbines and other complex mechanisms, as well as for Wall Street to project the price of commodities based on IoT data collected from farmers' combines in the Midwest. The latter is a great example of how the same data is being integrated and analyzed to fulfill several end users' needs.

05 The most common issue affecting Big Data projects is a **"lack of vision,"** though this was expressed in several ways by respondents. Lack of talent and security were also mentioned by multiple respondents. Companies are slow to see how Big Data can provide them with a business advantage and they tend to be vague about what they want to accomplish. Companies interviewed frequently serve as consultants, helping their clients understand what they can and cannot do with Big Data to address specific business problems and how to make the data they do have actionable for their business. Goals and expectations of the data quality can be unrealistic given the inconsistency of the data and the preparation required for analysis. Companies don't know what they don't know, and there is a lack of qualified talent in the market and in the enterprise. The skillset shortage is not going away. In addition, moving data around is inherently unsafe. You need someone who understands the infrastructure and the security protocols; however, these people are nearly as few and far between as Big Data professionals.

06 There was a consistent response regarding the future of Big Data—more data, faster, in more formats, from more sources, with faster analysis, as well as real-time integration and decision making to **solve problems before they occur**. Data is the oil of the 21st century. The innovation gap is shrinking. More businesses will focus on what you need to achieve to see an ROI on their Big Data initiatives. IoT will drive an order of magnitude increase in the amount of data collected and stored. As such, we'll need to decide on the fly what data to analyze, store, and throw away. While data is getting bigger and faster, we need to ensure security, governance, oversight, and policies are in place to protect the data and personal identifiable information (PII).

07 The biggest concerns around the state of Big Data today are: **1) privacy and security; 2) lack of collaboration and an ecosystem mentality; and, 3) the need to deliver business value and solve problems**. We need to establish standards so everyone involved is part of a Big Data ecosystem addressing clashes between protocols, data transfer, and disparate data sources. When we're connecting data across different entities, we need to ensure the connections are secure on both ends.

08 There is very little alignment with regards to what developers need to know to be successful working on Big Data projects. In general, developers need to **have traditional programming principles and skills that result in the creation of "rock-hard" applications, while remaining nimble and prepared for change, since it is inevitable.** The most recommended tools and languages are Java, SQL, Scala, Spark, C, R, and Python. Learn the ecosystems for the packages you control. Separate ingestion from data analytics and get more comfortable with data science. Ramp up statistics and applied math coding skills. Pick up statistics coding since it's the foundation of data science. Understand the architecture and how to build a system from the ground up that scales and handles large amounts of data. Lastly, go to the balcony and see how others are using the tools in an unbiased way. Make sure your end users are receiving value from the data you're providing.

09 Asking respondents what else they had on their mind regarding Big Data raised a diverse collection of questions and thoughts:

- As the next generation of tools come online, what do we need to keep in mind from an application perspective with regards to **cloud, scale, encryption, and security**?
- What are some **new use cases** given the capabilities of the platforms and tools? Give developers the opportunities to use all of the new extensions and see what they're able to come up with.
- How sophisticated and "bought in" are developers to Big Data?
- Where does the data science/predictive analytics world **align with business intelligence**?
- Systems will be able to handle the size, types, and velocity of data. **If analytical tools don't keep up with the changes they'll fall by the wayside**.
- **"Big Data" is simply replaced with "data."**
- **Where are people placing their Big Data** to do analytics—in the cloud, on premise, hybrid, local, global?
- **What about Blockchain?** It's on our radar since it will result in a sea change for economic transactions. It's the next big topic to be hyped.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

Machine Learning:

THE BIGGER PICTURE

QUICK VIEW

01 Always check if a problem is supervised or unsupervised beforehand.

02 The generative classifier contains additional likelihood information which the discriminative classifier does not have.

03 Non-parametric classifiers are often more effective when only a small training set is available.

BY **TAMIS ACHILLES VAN DER LAAN**
DATA SCIENCE AND MACHINE LEARNING SPECIALIST AT **STACKSTATE**

MACHINE LEARNING

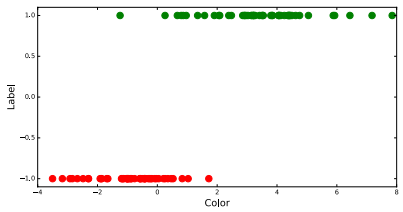
Machine learning is a field of computer science that concerns itself with the creation of algorithms that can learn from data and allows us to solve problems which cannot be programmed directly by hand such as face recognition. The fundamental idea is, instead of writing an algorithm that recognizes faces directly we write an algorithm that learns to recognize faces indirectly by example. The algorithm takes these examples and learns what is called a model which quantifies what constitutes a face or not. Because the system learns based off examples we can feed in samples in a continuous manner such that the algorithm can update its internal model continuously. This ensures we are always able to recognize faces even with changing trends in facial hair. The definition of machine learning is very broad, i.e. algorithms that learn from data and hence is often applied in context. Some examples include computer vision, speech recognition and natural language processing. Machine learning is often applied in conjunction with big data systems. For example for analyzing large volumes of text documents in order to extract there topic. In London a system is operational that tracks the movement of people using camera's all across the city. In the US a system is being developed which uses a single camera attached to a drone to observe whole city ad hoc (youtu.be/QGxNyaXfJsA).

MACHINE LEARNING BY EXAMPLE

In order to get an idea of how machine learning works we are going to use a fairly simple example. We are going to build an industrial sorting machine that needs to differentiate between tomatoes and kiwis. The industrial machine uses a special measuring device that uses a laser to measure the color of the object on the assembly line from red to green. Using this information the sorting machine has to decide to put object into the tomato or kiwi bin. A machine learning algorithm that allows us to differentiate between different classes is called a classifier.

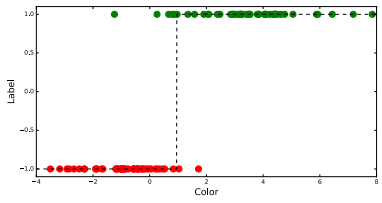
THE DISCRIMINATIVE CLASSIFIER

To start of our machine learning journey we need samples of tomatoes and kiwis: 50 different tomatoes and 50 different kiwis. We then use our measuring device to measure all our examples. This results into 100 float measurements that indicates the red to green color of the fruit, besides the measurements we also record in which class the fruit falls, namely the tomato or kiwi class. We will label the tomato class with a value of -1 and the kiwi class with a label of +1. We thus have two sets of 50 measurements and 50 labels. Let's plot our measurements on the x axis and the labels on the y axis.

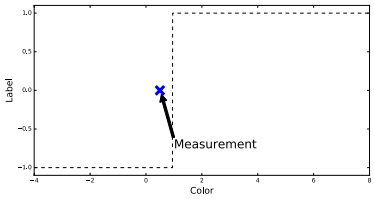


We see that the tomatoes and kiwis tend to lie in their own space on the horizontal axis. Even though there is some overlap between the two classes. This is to be expected as there exist for example unripe tomatoes that have a greenish color. This also tells us that only measuring color is probably not enough information to properly separate all kiwis and tomatoes, but for demonstration purposes I have chosen to keep things simple. If we take a measurement of a unknown fruit we can now classify it as either a tomato or kiwi by simply looking in which region the fruit is located. What we have constructed is called discriminative classifier. The discriminative part refers to the

fact it splits space into regions corresponding to classes. This type of classifier has been very successful and is used all over in many applications. The next step in the process is to build a model that allows us to separate between the two fruits. We do this by setting up a function that is -1 for the tomato region and +1 for the kiwi region and consequently decides space into two class regions.

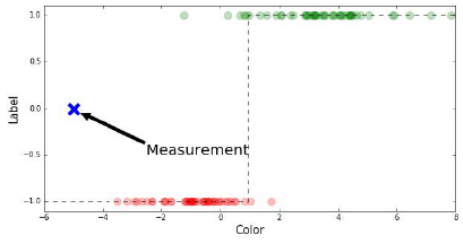


The boundary between the two class regions is called the classification boundary. The goal is to divide the space in such a way that as much fruit is labeled as correctly as possible by our model. This process is called learning and we will talk about this process later on in the article. Given our model we can now in essence throw away our samples and simply apply the model to classify new objects as either a tomato or kiwi. We do this by measuring the color of the object and deciding in which class region this measurement falls.

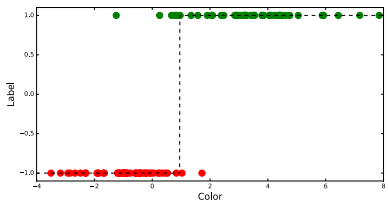


In the example above we see that our measurement falls into the -1 region and we thus label the object as tomato. This approach is fine but as one caveat associated with it. Our model does not retain any information about the density or frequency of our data with respect to our measurement. To explain what this means consider the following scenario, what if our new sample lies very far away from all our training examples.

The question now is, are we looking at an exceptionally red tomato, or has something gone wrong, for example a piece of brightly colored red plastic has found its way into the system? This is a question our discriminative classifier can't answer.



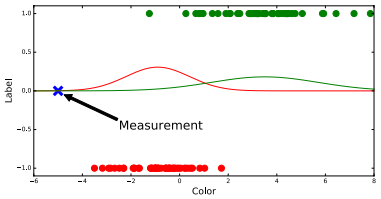
THE GENERATIVE CLASSIFIER



In order to deal with the limitation stated above we introduce the generative classifier. Our generative classifier will make use of two normal distributions in order model our two fruit classes. Each normal distribution will represent the probability of the class given the color measurement. That is each normal distribution models the likelihood of a measurement being part of its respective class.

In order to dis3 criminate between tomatoes and kiwis we now look at which of the two probability density functions is larger in order to classify a new measurement. In fact we can again split the space into two regions as before and throw away the normal distributions all together if we wanted to.

This is something we will not do, because the normal distributions include extra information we can use. Namely the normal distributions can tell us if a sample is likely to belong to one of the classes or not.



Consider the measurement from figure 4 which falls far away from the samples in our training set. We see that the measurement is not only improbable according to our training samples but also according to our model, i.e. the normal distributions. Hence we could detect this with our model and stop the assembly line for inspection by an operator. You might wonder where the word generative comes from. The word generative comes from the fact that we can draw samples according to our normal distributions and in effect generate measurements.

LEARNING AND OPTIMIZATION

So far we have seen that there are two classes of machine learning algorithms, discriminative and generative. We assume that we have a set of samples (actually called the training set) with which we can construct our model. Next we will take a look at how we actually construct a model using the training data, the process of which is called learning.

Let's start with the generative classifier, we are looking to fit two normal distributions to our data. The two normal distributions are quantified using their mean μ and standard deviation σ. So if we find those parameters we have constructed our model and fitted the normal distribution onto their respective class data. To compute the means we take the sample average of our measurements in each class. To compute the standard deviation we take the square root of the sample variance in the class data. The sample mean and deviation can be computed using the following two equations:

$$\bar{\mu} = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$\bar{\sigma} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{\mu})^2}$$

Next we move onto the discriminative classifier. This can be illustrated with our discriminative classifier. Our discriminative classifier uses the following function as its model:

$$f(x) = sign[a \cdot x + b]$$

We have measurement x, constants a and b that determine where the measurement will be separated into the –1 and +1 regions. The goal is to find values for a and b such that as many samples are classified correctly.

Finding the parameters is not always analytically tractable, this is also the case for our discriminative 4 classifier, in this case

we require some kind of optimization method which finds the parameters for us iteratively. We use the following optimization algorithm to find the values of a and b:

ALGORITHM 1: *OPTIMIZATION ALGORITHM*

1. Start with a random guess for parameters *a* and *b*.

2. Update variables for all samples with measurement x and label y:

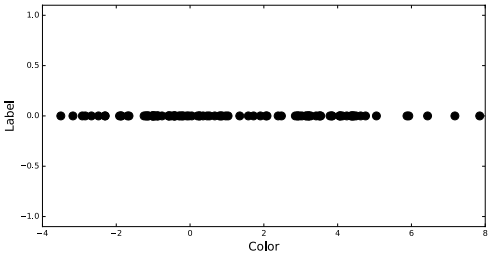
$$a \leftarrow a + \alpha \cdot [f(x) - y]$$
$$b \leftarrow b + \alpha \cdot [f(x) - y]$$

3. If maximum iterations not met go back to step 2.

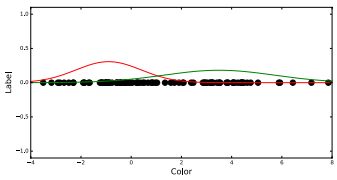
In this algorithm *f(x)* is the result of applying our model to sample measurement x. This produces a –1 or +1 classification. We then compare this result to the original label y associated with the sample. We then update the parameters based on this comparison, i.e. we move the decision boundary in such a way the sample is more likely to be updated. The size of the update is defined by α which is called the learning rate.

SUPERVISED AND UNSUPERVISED -LEARNING

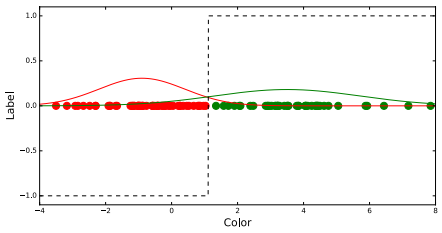
Now that we know the difference between a generative and discriminative classifier as well as how to learn the parameters of such a classifier we now move on to unsupervised learning. You may not have realized it but in the example above we have applied supervised learning. We had a training set consisting of our measurements (fruit color) and labels (tomato or kiwi). But what if we don't know the label of our examples? In other words we have measurements of fruit we know are either of the type tomato or kiwi but we don't know which.



This is the problem of unsupervised learning, and is considered more complex then supervised learning. The way to tackle this problem is to refer back to our generative method of classification. Using our generative method we fitted two gaussians onto our class data. This time however we have no class data and need to fit our gaussians to our data without knowledge of our classes. The way we can do this is by trying to fit the two gaussians such that they explain our data as good as possible. We want to maximize the likelihood that our data was generated by our two normal distributions. The way to do this is by using a fairly complex optimization algorithm called expectation maximization which is an iterative procedure for finding the maximum a posteriori estimates of the parameters. This algorithm lies beyond the scope of this article but many resources on the subject exist. After applying expectation maximization optimization algorithm we get the values of the means and standard deviations producing the following result:

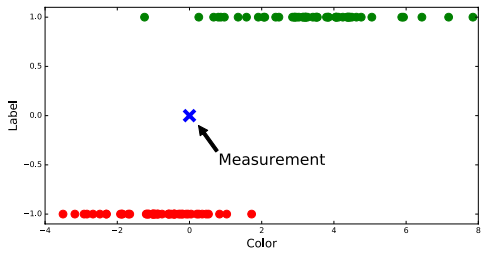


Now we can classify points by comparing the likelihoods as before.



PARAMETRIC AND NON-PARAMETRIC ALGORITHMS

Next we are going to look at the difference between parametric and non-parametric classifiers. The term parametric algorithm refers to the parameters your model uses to define your model. Using a parametric algorithm we use the training data to extract values for our parameters that completely define our model. The two models we have looked at so far both use parameters. The discriminative model uses two parameters a and b, while the generative model uses four parameters two means $\bar{\mu}_1$, $\bar{\mu}_2$ and two standard deviations $\bar{\sigma}_1$, $\bar{\sigma}_2$. After finding these parameters we no longer need the training set and can literally throw it away. Non parametric algorithms don't require any parameters but instead use the training set directly. Non parametric algorithms don't use parameters to specify there model but instead use the training data directly. This often results in more computational overhead and memory usage. But these non parametric classifiers are often more accurate and easy to set up on small training sets compared to parametric classifiers.



A good example of a non-parametric algorithm is the k nearest neighbor classifier. Given a new measurement we look up the *k* nearest neighbors and assign the most frequent class label to the measurement. When we take the nearest 5 neighbors of our measurement and look at our labels we get -1,-1,-1,+1,-1. We see that -1 is the most frequent label and hence we correctly assign our measurement to the -1 i.e. tomato class.

TAMIS ACHILLES VAN DER LAAN is a Data Science and Machine Learning specialist focused on the design of intelligent learning systems. He has a passion for applied math and likes to think his way through problems. Tamis works for StackState (stackstate.com), which produces a real-time IT operations platform for medium and large IT organizations. His work is focused on anomaly detection methods for identifying components in the IT landscape that have failed or are about to fail.

Solutions Directory

The big data solutions landscape is big. So we've picked 156 of the best tools, platforms, and frameworks for data ingestion, storage, querying, analytics, graph processing, resource management, warehousing, visualization, and more. Check them out below and let us know what we've missed.

PART I: COMMERCIAL PRODUCTS

PRODUCT	COMPANY	DESCRIPTION	WEBSITE
Actian Analytics Platform	Actian	Analytics for Hadoop and MPP (fast and OLAP); multi-model (relational, object) and multi-platform (PSQL for embedded) DBMS; RAD platform	actian.com/products/big-data-analytics-platforms-with-hadoop
Alation	Alation	Enterprise data collaboration platform	alation.com
Alpine Chorus 6	Alpine Data	Data science, Machine Learning, ETL, execution workflow design and management	alpinedata.com/product
Alteryx Analytics Platform	Alteryx	Low/no-code ETL, predictive analytics (GUI is generated and editable R), spatial analytics, reporting and visualization integrations, analytics server, cloud analytics app platform	alteryx.com
Amazon Kinesis	Amazon Web Services	Stream data ingestion, storage, query, and analytics PaaS	aws.amazon.com/kinesis
Amazon Machine Learning	Amazon Web Services	Machine Learning algorithms as a service, ETL for ML, data visualization and exploration, modeling and management APIs, batch and realtime predictive analytics	aws.amazon.com/machine-learning
Attunity Big Data Management	Attunity	Data warehousing and ETL; movement, preperation, and usage analytics	attunity.com
BI Office	Pyramid Analytics	Data discovery and analytics platform	pyramidanalytics.com/pages/bi-office.aspx
BigML	BigML	Machine Learning REST API, analytics server, development platform	bigml.com
Birst	Birst	Enterprise and embedded BI and analytics platform	birst.com
Bitam Artus	Bitam	BI platform	bitam.com
BOARD All in One	Board	BI, analytics, corporate performance management platform with no-code tools	board.com
Capsenta	CAPSENTA	Database wrapper (for lightweight data integration)	capsenta.com
Cask	Cask Data	Containers (data, programming, application) on Hadoop for data lakes	cask.co
Cask Data App Platform	Cask Data	Analytics platform for YARN with containers on Hadoop, visual data pipelining, data lake metadata management	cask.co
Cirro	Cirro.com	Data warehousing on cloud-enabled information fabric	cirro.com

PRODUCT	COMPANY	DESCRIPTION	WEBSITE
Cisco Big Data and Analytics	Cisco	Edge analytics, data virtualization, data preparation	cisco.com/c/en/us/products/analytics-automation-software/data-analytics-software
Cloudera Enterprise	Cloudera	Comprehensive Big Data platform on enterprise-grade Hadoop distribution	cloudera.com
Cortana Intelligence Suite	Microsoft	Data warehousing, data lake, structured and unstructured data stores, Machine Learning implementations, batch and streaming services (Hadoop, Spark, Microsoft R Server, HBase, Storm), BI, bot framework, various cognitive APIs (vision, speech, language, semantic, search)	azure.microsoft.com/en-us/services/machine-learning
Databricks	Databricks	Data science (ingestion, processing, collaboration, exploration, visualization) on Spark	databricks.com
DataDirect	Progress Software	Data integration: many-source, multi-interface (ODBC, JDBC, ADO.NET, OData), multi-deployment	progress.com/datadirect-connectors
Dataguise	Dataguise	Big Data security	dataguise.com
Datameer	Datameer	BI, data integration and ETL, visualization on Hadoop	datameer.com
DataRPM	DataRPM	Cognitive predictive maintenance for industrial IoT	datarpm.com
DataTorrent RTS	DataTorrent	Stream and batch platform (based on Apache Apex), data management, GUI app dev, visualization and reporting	datatorrent.com
DataWatch	DataWatch	Data extraction and wrangling, self-service analytics, streaming visualization, server	datawatch.com
EngineRoom	EngineRoom.io	Geospatial, data transformation and discovery, modeling, analytics, visualization	engineroom.io
Exaptive	Exaptive	RAD and application marketplace for data science	exaptive.com
Exasol	Exasol	MPP in-memory database for analytics	exasol.com
FICO Decision Management Suite	FICO	Decision management platform	fico.com/en/analytics/decision-management-suite
GoodData Platform	GoodData	Data distribution, visualization, analytics (R, MAQL), BI, warehousing	gooddata.com
Hortonworks Data Platforms	Hortonworks	Comprehensive Big Data platform on enterprise-grade Hadoop distribution	hortonworks.com
H2O	H2O.ai	Data science (R, Python, Java, Scala) and deep learning on Spark with model scoring, unified wrapper API	h2o.ai
HP Haven	Hewlett Packard Enterprise	Data integration, analytics (SQL and unstructured), exploration, Machine Learning	www8.hp.com/us/en/software-solutions/big-data-analytics-software.html
IBM InfoSphere	IBM	Data integration, management, analytics on Hadoop, data warehousing, streaming	www-03.ibm.com/software/products/en/category/bigdata
Infobright Enterprise	Infobright	Column-oriented store with semantic indexing and approximation engine for analytics	infobright.com
Informatica	Informatica	Data management, integration, lake, prep, analytics, BPM	informatica.com
Information Builders	Information Builders	BI and analytics	informationbuilders.com

PRODUCT	COMPANY	DESCRIPTION	WEBSITE
Insights Platform	1010data	Data management, analysis, modeling, reporting, visualization, RAD apps	1010data.com/products/insights-platform/analysis-modeling
Jaspersoft	Tibco	BI, analytics (OLAP, in-memory), ETL, data integration (relational and non-relational), reporting, visualization	jaspersoft.com
Kapow	Kofax	Data extraction, integration, BI	kofax.com
Kognitio Analytical Platform	Kognitio	In-memory, MPP, SQL and NoSQL analytics on Hadoop	kognitio.com
Liaiason Alloy	Liaison Technologies	Data management and integration	liaison.com
Logentries	Rapid7	Log management and analytics	logentries.com
Loggly	Loggly	Cloud log management and analytics	loggly.com
Logi	Logi	Embedded BI, data discovery	logianalytics.com
MapR Converged Data Platform	MapR	Comprehensive Big Data platform on enterprise-grade Hadoop distribution with many integrated open-source tools (Spark, Hive, Impala, Solr, etc.), NoSQL (document and wide column) DBMS	mapr.com
Microsoft Power BI	Microsoft	Business intelligence	powerbi.microsoft.com
MicroStrategy	MicroStrategy	Data management, analytics, BI	microstrategy.com
Necto	Panorama Software	Business Intelligence, Consulting	panorama.com
New Relic Insights	New Relic	Real-time application performance analytics	newrelic.com/insights
OpenText Big Data Analytics	OpenText	Analytics and visualization (GUI->code) with analyatics server	opentext.com/what-we-do/products/analytics/opentext-big-data-analytics
Oracle Big Data Management	Oracle	Hadoop as a service, Big Data appliances, connectors (Oracle DB with Hadoop), in-memory data grid, key-value NoSQL DBMS	oracle.com/big-data
Palantir Gotham	Palantir	Cluster data store, on-the-fly data integration, search, in-memory DBMS, ontology, distributed key-value store	palantir.com/palantir-gotham
Paxata	Paxata	Data integration, preparation, exploration, visualization on Spark	paxata.com
Pentaho	Pentaho	Data integration layer for Big Data analytics	pentaho.com
Pivotal Big Data Suite	Pivotal	Data warehousing, SQL on Hadoop, in-memory data grid, data ingestion, Machine Learning	pivotal.io
Platfora	Platfora	Data discovery and analytics on Hadoop and Spark	platfora.com
Prognoz Platform	Prognoz	BI and analytics (OLAP, time series, predictive)	prognoz.com
Qlik Platform	Qlik	BI (with self-service), analytics, visualization, data market	qlik.com
Qubole	Qubole	Data engines for Hive, Spark, Hadoop, Pig, Cascading, Presto on AWS, Azure, Google Cloud	qubole.com

PRODUCT	COMPANY	DESCRIPTION	WEBSITE
RapidMiner Studio	RapidMiner	Predictive analytics on Hadoop and Spark with R and Python support, server, cloud hosting, plugins	rapidminer.com
RedPoint Data Management	RedPoint	Data management, quality, integration (also on Hadoop)	redpoint.net
RJMetrics	RJMetrics	Analytics for e-commerce	rjmetrics.com
SAP HANA	SAP	In-memory, column-oriented, relational DBMS (cloud or on-premise) with text search, analytics, stream processing, R integration, graph processing	hana.sap.com
SAS Intelligence Platform	SAS	Analytics, BI, data management, deep statistical programming	sas.com
Sisense	Sisense	Analytics, BI, visualization, reporting	sisense.com
Skytree Infinity	Skytree	Machine Learning platform with self-service options	skytree.net
Splunk Enterprise	Splunk	Operational intelligence for machine-generated data	splunk.com
Spring XD	Pivotal	Data ingestion, real-time analytics, batch processing, data export on Hadoop, Spark, and any data store	projects.spring.io/spring-xd
Stitch	Stitch	ETL as a service	stitchdata.com
Sumo Logic	Sumo Logic	Log and time-series management and analytics	sumologic.com
Tableau Desktop	Tableau	Visualization, analytics, exloration (with self-service, server, hosted options)	tableau.com
Talend	Talend	Real-time data integration, preparation, fabric	talend.com
Targit	Targit	BI, analytics, discovery front-end with self-service options	targit.com
Teradata	Teradata	Data warehousing, analytics, lake, SQL on Hadoop and Cassandra, Big Data appliances, R integration, workload management	teradata.com
Terracotta In-Memory Data Management by Software AG	Software AG	In-memory data management, job scheduler, Ehcache implementation, enterprise messaging	terracotta.org
ThingSpan	Objectivity	Graph analytics platform with Spark and HDFS integration	objectivity.com/products/thingspan
TIBCO Spotfire	TIBCO	Data mining and visualization	spotfire.tibco.com
Treasure Data	Treasure Data	Analytics infrastructure as a service	treasuredata.com
Trifacta Platform	Trifacta	Data wrangling, exploration, visualization on Hadoop	trifacta.com
Waterline Data	Waterline Data	Data marketplace (inventory, catalogue with self-service) on Hadoop	waterlinedata.com
Yellowfin	Yellowfin	Business Intelligence, Data Visualization	yellowfinbi.com
Zaloni	Zaloni	Enterprise data lake management	zaloni.com
Zoomdata	Zoomdata	Analytics, visualization, BI with self-service on Hadoop, Spark, many data stores	zoomdata.com

SOLUTIONS DIRECTORY **PART II: OPEN-SOURCE FRAMEWORKS**

Many popular open-source frameworks have grown up around Hadoop, many of which are managed by the Apache Software Foundation. The most commonly used open-source big data frameworks are listed below.

PROJECT	DESCRIPTION	WEBSITE
Alluxio (formerly Tachyon)	Distributed storage system across all store types	alluxio.org
Ambari	Hadoop cluster provisioning, management, and monitoring	ambari.apache.org
Apex	Stream+batch processing on YARN	apex.apache.org
Avro	Data serialization system (data structure, binary format, container, RPC)	avro.apache.org
Beam	Language-specific SDKs for defining and executing dataflows for parallelization	beam.incubator.apache.org
BIRT	Visualization and reporting library for Java	eclipse.org/birt
Cascading	Abstraction to simplify MapReduce and dataflow coding on Hadoop and Flink	cascading.org
Ceph	Distributed object and block store and file system	ceph.com
Chart.js	Simple JavaScript charting library	chartjs.org
Crunch	Java library for writing, testing, running MapReduce pipelines	crunch.apache.org
D3.js	Declarative-flavored JavaScript visualization library	d3js.org
Disco	MapReduce framework esp. for Python	discoproject.org
Drill	Distributed queries on multiple data stores and formats	drill.apache.org
Druid	Columnar distributed data store w/realtime queries	druid.io
Falcon	Data governance engine for Hadoop clusters	falcon.apache.org
Flink	Streaming dataflow engine for Java	flink.apache.org
Flume	Streaming data ingestion for Hadoop	flume.apache.org
GFS	(Global File System) Shared-disk file system for Linux clusters	git.kernel.org/cgit/linux/kernel/git/gfs2/linux-gfs2.git/?h=for-next
Giraph	Iterative distributed graph processing framework	giraph.apache.org
GraphViz	Graph(nodes+edges) visualization toolkit	graphviz.org
GraphX	Graph and collection processing on Spark	spark.apache.org/graphx

PROJECT	DESCRIPTION	WEBSITE
GridMix	Benchmark for Hadoop clusters	hadoop.apache.org/docs/r1.2.1/gridmix.html
H2O	Stats, machine learning, math runtime for big data	h2o.ai
Hadoop	MapReduce implementation	hadoop.apache.org
Hama	Bulk synchronous parallel (BSP) implementation for big data analytics	hama.apache.org
HAWQ	Massively parallel SQL on Hadoop	hawq.incubator.apache.org
HDFS	Distributed file system (Java-based, used by Hadoop)	hadoop.apache.org
Hive	Data warehousing framework on YARN	hive.apache.org
Ignite	In-memory data fabric	ignite.apache.org
Impala	Distributed SQL on YARN	impala.io
InfoVis Toolkit	JavaScript visualization library	philogb.github.io/jit
Java-ML	Various machine learning algorithms for Java	java-ml.sourceforge.ne
Julia	Dynamic programming language for scientific computing	julialang.org
JUNG	Graph(nodes+edges) framework (model, analyze, visualize) for Java	
Jupyter	Interactive data visualization and scientific computing on Spark and Hadoop	jupyter.org
Kafka	Distributed pub-sub messaging	kafka.apache.org
MADlib	Big data machine learning w/SQL	madlib.incubator.apache.org
Mahout	Machine learning and data mining on Hadoop	mahout.apache.org
Matplotlib	Plotting library on top of NumPy (like parts of MATLAB)	matplotlib.org
Mesos	Distributed systems kernel (all compute resources abstracted)	mesos.apache.org
Misco	MapReduce Framework	alumni.cs.ucr.edu/~jdou/misco
NumPy	Mathematical computing library (multi-dimensional arrays, linear algebra, Fourier transforms, more) for Python	numpy.org
Oozie	Workflow scheduler (DAGs) for Hadoop	oozie.apache.org
OpenTSDB	Time-series database on Hadoop	opentsdb.net
ORC	Columnar storage format	orc.apache.org
Pandas	Data analysis and modeling for Python	pandas.pydata.org
Parquet	Columnar storage format	parquet.apache.org

PROJECT	DESCRIPTION	WEBSITE
Phoenix	SQL->HBase scans->JDBC result sets	phoenix.apache.org
Pig	Turns high-level data analysis language into MapReduce programs	pig.apache.org
Pinot	Realtime OLAP distributed data store	github.com/linkedin/pinot
Presto	Distributed interactive SQL on HDFS	prestodb.io
Protocol Buffers	Data serialization format & compiler	developers.google.com/protocol-buffers/docs/overview
R	Language and environment for statistical computing and graphics	r-project.org
Samza	Distributed stream processing framework	samza.apache.org
SciPy	Scientific computing ecosystem (multi-dimensional arrays, interactive console, plotting, symbolic math, data analysis) for Python	scipy.org
Spark	General-purpose cluster computing framework	spark.apache.org
Spark Streaming	Discretized stream processing with Spark's RDDs	spark.apache.org/streaming
Sqoop	Bulk data transfer between Hadoop and structured datastores	sqoop.apache.org
SSRS	SQL Server reporting (server-side)	msdn.microsoft.com/en-us/library/ms159106.aspx?f=255&MSPPError=-2147217396
Storm	Distributed realtime (streaming) computing framework	storm.apache.org
Tableau	Interactive data visualization for BI	tableau.com
TensorFlow	Computation using dataflow graphs (nodes are math operations, edges are tensors between operations)	tensorflow.org
Tez	Dataflow (DAG) framework on YARN	tez.apache.org
Theano	Python library for multi-dimensional array processing w/GPU optimizations	deeplearning.net/software/theano
Thrift	Data serialization framework (full-stack)	thrift.apache.org
Weka	Machine learning and data mining for Java	cs.waikato.ac.nz/ml/weka
Wolfram Language	Knowledge-based programming language w/many domain-specific libraries	wolfram.com/language
YARN	Resource manager (distinguishes global and per-app resource management)	hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html
YCSB	General-purpose benchmarking spec	github.com/brianfrankcooper/YCSB/wiki/Getting-Started
Zeppelin	Interactive data visualization	zeppelin.apache.org
Zookeeper	Coordination and State Management	zookeeper.apache.org

GLOSSARY

ALGORITHM A series of instructions used to solve a mathematical problem.

APACHE HADOOP An open-source tool to process and store large distributed data sets across machines by using MapReduce.

APACHE SPARK An open-source Big Data processing engine that runs on top of Apache Hadoop, Mesos, or the cloud.

ARTIFICIAL INTELLIGENCE The ability of a machine to recognize its environment, act in a rational way, and maximize its ability to solve problems.

BIG DATA A common term for large amounts of data. To be qualified as big data, data must be coming into the system at a high velocity, with large variation, or at high volumes.

CLUSTER A subset of data that share particular characteristics. Can also refer to several machines that work together to solve a single problem.

DATA PROCESSING The process of retrieving, transforming, analyzing, or classifying information by a machine.

DATA VALIDATION The act of examining data sets to ensure that all data is clean, correct, and useful before it is processed.

GRAPH ANALYTICS A way to organize and visualize relationships between different data points in a set.

K-MEANS CLUSTERING ALGORITHM A method used to separate sets of data points into subsets called clusters, organized around nearby means.

LINEAR CLASSIFIER A piece of software used to sort variables by characteristics in machine learning by using linear combinations of those characteristics.

MACHINE LEARNING An AI that is able to learn by being exposed to new data, rather than being specifically programmed.

MAPREDUCE A data processing model that filters and sorts data, called the Map stage, then performs a function on that data and returns an output in the Reduce stage.

MESSAGE-ORIENTED MIDDLEWARE Software that transmits and receives messages sent between individual parts of a distributed system.

NEURAL NETWORK A system modeled on the brain and nervous system of humans, where processors operate parallel to each other and are organized in tiers, and each processor receives information from the processor preceding it in the system.

NEURON A cell that transmits information through electrical and chemical signals in the human brain. Processors in neural networks are based how on these cells operate.

NONDETERMINISTIC FINITE AUTOMATION (NFA) A finite state machine that can be in multiple states at once.

NORMAL DISTRIBUTION A common graph representing probability of a large number of random variables, where those variables approach normalcy as the data set increases in size. Also called a Gaussian distribution or bell curve.

NOSQL DATABASE Short for "not only SQL", or any database that uses a system to store and search data other than just using tables and structured query languages.

PARSE To divide data, such as a string, into smaller parts for analysis.

PERCEPTRON An algorithm that allows linear classifiers to learn, which forms the basis of one of the first artificial neural networks.

PERSISTENT STORAGE A non-changing place, such as a disk, where data is saved after the process that created it has ended.

REAL-TIME STREAM PROCESSING A model for analyzing sequences of data by using machines in parallel, though with reduced functionality.

RECURSION The act of a function calling on itself. A common example is the Fibonacci sequence.

REGULAR EXPRESSIONS A string that is used to create a specialized search pattern, usually to find and replace strings in a file. Also called regexes.

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS) A system that manages, captures, and analyzes data that is grouped based on shared attributes called relations.

RESILIENT DISTRIBUTED DATASET The primary way that Apache Spark abstracts data, where data is stored across multiple machines in a fault-tolerant way.

STRUCTURED QUERY LANGUAGE (SQL) The standard language used to retrieve information from a relational database.

TRANSFORMATION The conversion of data from one format to another.

VISUALIZATION The process of analyzing data and expressing it in a readable, graphical format, such as a chart or graph.