Actian DataFlow

Accelerate Analytic Data

A Technical Overview

Contents

Executive summary

Hadoop holds tremendous promise for large scale data management and data analytics projects that could be of huge benefit to many enterprises. However, Hadoop has limitations and difficulties of use that cause many projects to fail. These include the need for rare and expensive skillsets, inadequate execution speed, long implementation cycles, and extreme difficulty of incorporating other datasets. Actian DataFlow solves these challenges with a scale-up, scale-out architecture, automatic workload optimization, pipeline parallelism, and a wide range of pre-built operators in a easy-to-use, visual interface. Actian DataFlow provides unmatched price/performance and due to platform agnosticism, fits into most enterprise architectures with ease.

The Promise and Challenge of Hadoop

> *"Hadoop is a component, not a solution."*
>
> – *Robin Bloor, The Bloor Group*

In the modern world, dozens of new data sources are coming at the corporate world faster than the technology can keep up. Hadoop has been at the heart of many modernization efforts. Hadoop offers unlimited scalability, the ability to handle virtually any kind of data, regardless of structure, and a price/performance level that is orders of magnitude better than previous data management technologies. But, Hadoop has five drawbacks that have hampered its adoption in many enterprises, and even caused the failure of what should have been high ROI analytics projects.

1. Hadoop requires highly skilled developers with a deep understanding of the complexities of building parallel programs, specifically in the MapReduce paradigm. The difficulty level of parallel and distributed programming means that these human resources have proven to be scarce, very expensive, and difficult to either find or train.

2. The implementation cycle, the time between identifying a business problem that Hadoop data might solve and when actionable outputs can be reached, is often prohibitively long. The skillsets needed to design good analytics or any good business solutions that use the data stored on Hadoop, and the skillsets needed to work on Hadoop are rarely found in the same people. It can take a large team, a lot of back and forth iteration cycles, and a lot of coding time to come up with a usable solution. By then, the business may have moved on, and the answer may not even be useful anymore.

3. The performance speed of data crunching activities on Hadoop are surprisingly slow. Hadoop provides a tremendous leap forward in price/performance over traditional data warehouse and appliance

models, as well as unlimited scalability, but the MapReduce parallel programming paradigm was only designed to do specific tasks well, such as text search. Once you move beyond the specific tasks it was designed for into more sophisticated data processing, it bogs down enough that response times become unacceptable for most business applications.

4. Enhancing Hadoop data with other data sources is prohibitively difficult. Even in modern enterprises that have embraced this new technology, the projects done on Hadoop are often done in isolation from the rest of the enterprise. Joining and enhancing data stored in Hadoop with other sources is rarely even considered. MapReduce was only designed to process Hadoop data.

5. Inconsistencies across Hadoop distributions. Individual Hadoop distributions each have their own approach to providing SQL access to Hadoop data. They all have performance and scalability challenges to varying degrees, don't fully support the SQL standards (although they are getting better), and don't support efficient singleton updates for operational workloads.

Accelerating Hadoop with Actian

Actian DataFlow™ works in a complementary way with Vector to address those weaknesses. DataFlow is a critical component for improving data load times, and for orchestrating, managing, and optimizing query execution that takes maximum advantage of Vector's fast processing capabilities. It does this without requiring any parallel programming expertise at all, Actian DataFlow speeds data processing on Hadoop clusters from 2X to 100X, or more, and vastly reduces implementation cycles from end to end, often saving businesses several months. Actian DataFlow provides three main advantages over working with Hadoop directly via MapReduce, or other intermediate code generators.

First, Actian automatically accelerates data processing execution speed on Hadoop, without requiring additional skills or additional work of any kind. This speed of execution allows preparation and processing of virtually unlimited volumes of data in reasonable timeframes – removing the need to take small samples for the sake of the software, not the science. This allows for more in-depth analyses with greater degrees of freedom, and higher levels of confidence in the accuracy of results.

Secondly, Actian enhances Hadoop data with data from a wide variety of other sources. High speed connectivity options to nearly any data source and the ability to join heterogeneous data allows analysis of multiple data sources simultaneously. This provides the data needed for greater context in business solutions, providing that 360-degree view that can be so valuable. This can provide more meaningful results, and often much higher lift from actions taken

from the analyses of that data.

And finally, **Actian vastly shortens the time-to-value in analytics projects.** Various aspects of Actian DataFlow, from the drag-and-drop simplicity of the interface to the automatic data flow based parallel optimization of the underlying framework shorten the entire data management and analytics process eliminating each choke point that slows down traditional approaches. This provides a far faster path from the moment a new analytics need is identified, to the point when it can be acted on to improve the business outcome.

By letting you use ALL of your Hadoop data, Actian provides higher accuracy. By allowing you to use ALL TYPES of data in your enterprise environment, Actian provides better analytics context. And by addressing all the worst chokepoints in the analytics process from end to end, Actian provides the best time-to-value possible.

Automate Execution Optimization

"Analytical latency reduction is Actian's number one advantage. Speed equals value as long as you get the answers right."

– *Robin Bloor, The Bloor Group*

Actian DataFlow automatically optimizes workflows for extremely high execution speed on Hadoop clusters or any multi-core or cluster environment. DataFlow does this by making maximum use permitted of all available hardware. As hardware potential is increased, performance speed also increases at a near-linear scale. This does not require the user to know anything about parallel programming. The underlying DataFlow engine framework automatically handles the parallelization and optimization aspect.

Understanding how DataFlow automatically optimizes parallel data processing is the key to understanding how DataFlow achieves the highest price/performance possible.

Scale up, Scale out

The roots and patented IP of Actian DataFlow pre-date the cluster computing revolution. When DataFlow, then known as DataRush, was first begun, the goal of the project was to make optimum use of the new multi-core style of computers. This was about the time that dual-core machines were starting to become the standard. So, originally, DataFlow was designed to scale up on a single computer, taking full advantage of all cores and threads available on a

single machine up to a settable limit, whether that was 2 or 256.

The focus of development efforts was on gaining near-linear scaling. If a program was executed on a 4-core server in 10 seconds, that same program should, without any alteration, execute on an 8-core server in approximately 5 seconds and a 16-core server in approximately 2.5 seconds. This goal was achieved.

Later, as it became clear that Hadoop and distributed cluster computing would soon become the standard for handling large datasets, the ability for DataFlow-based applications to scale out to multiple nodes on a cluster was added. When YARN first began to be available, the ability to share resources on a Hadoop cluster through that channel was immediately added.

This history means that DataFlow will make maximum use of each individual node on a cluster, and then make optimum use of all the nodes on a cluster. Because DataFlow is thread parallel at its base, this makes it more efficient than process-based parallelism such as MapReduce. Dataflow will always provide better machine utilization at the node level. DataFlow will both scale up and scale out.

The answers to, “How does it do that?” are below.

Automatic runtime thread availability detection

The first contributor to both Actian DataFlow’s simplicity of development and execution speed is the built-in automatic detection of available compute power. As mentioned above, in the early days of DataFlow’s development, DataFlow’s purpose was to take advantage of the new multi-core servers. Since, there was no telling how many cores would eventually become the norm, one of the first capabilities built into DataFlow was the ability to detect hardware core and thread availability. The ability to then expand that to detect all cores, threads, and nodes available in a cluster was later added without much difficulty since the two capabilities are very similar.

This automatic compute power availability assessment is done **at runtime**. This may not seem that important at first, but it means that a workflow only needs to be developed once, and will always be optimized for its particular environment in a “just in time” manner. The design can be created or tested on a 4-core laptop, a 16-core server, or a 3000-node cluster. That same design can be deployed on any of those environments, without any re-design, tweaking, or other performance optimization work. And, in a year or two, if the 3000-node cluster becomes a 6000-node cluster with the new nodes each having double the cores of the old nodes, the workflow will simply, when the application is run, detect the increase in hardware availability and execute that much faster.

The next logical question is, once DataFlow auto-detects hardware power availability at runtime, how does it then take advantage of it without anyone having to modify the application?

Data flow design principals

Part of the power of Actian DataFlow is that it is based on a data programming paradigm known, not too surprisingly, as data flow. This concept pre-dates the Actian DataFlow framework, and is used in many other applications, such as high-performance computing.

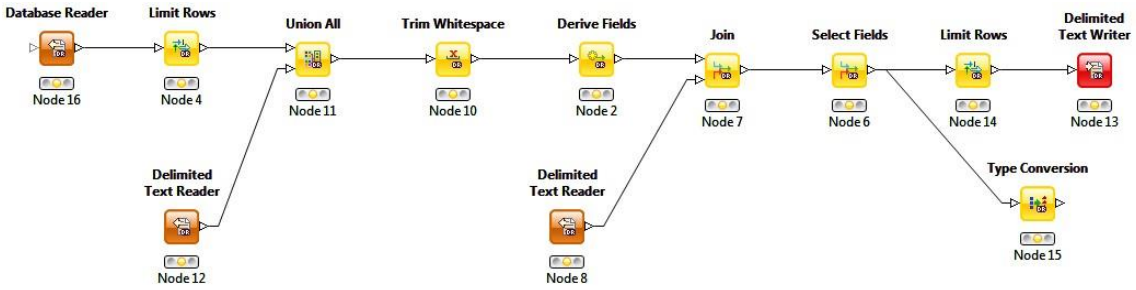
The data flow programming paradigm essentially conceives of all applications as a series of nodes and edges. To avoid confusion with computers in a cluster which are also called nodes, I'll refer to data flow nodes as operators. The operators are units of data processing that start with one or more data inputs and ends with one or more data outputs. Each operator performs one specific task, such as reading a particular type of data from disk to memory, transforming one data type to another, joining one dataset with another, evaluating statistics such as mean or range, or finding unique values in a column. These individual functions are not limited to the basic tasks like a mapper or reducer. Even a sophisticated algorithm such as a decision tree machine learner or predictor can be a single DataFlow operator.

Each one of these operators represents a function in an end-to-end application sequence called a data flow or Directed Acyclic Graph (DAG). I generally refer to this as a workflow to avoid technology specific terms. An Actian DataFlow workflow usually begins with access to a data source, such as HDFS or HBase or an RDBMS, and ends with some sort of output. Whether that output is more data, a PMML file that can be passed to another application for additional analysis, or some sort of prediction, or automated action, that is entirely up to the person building the workflow.

The edges are the connections between the output of one operator and the input of the next. They determine what happens first, next, etc. and how the different operators relate to each other.

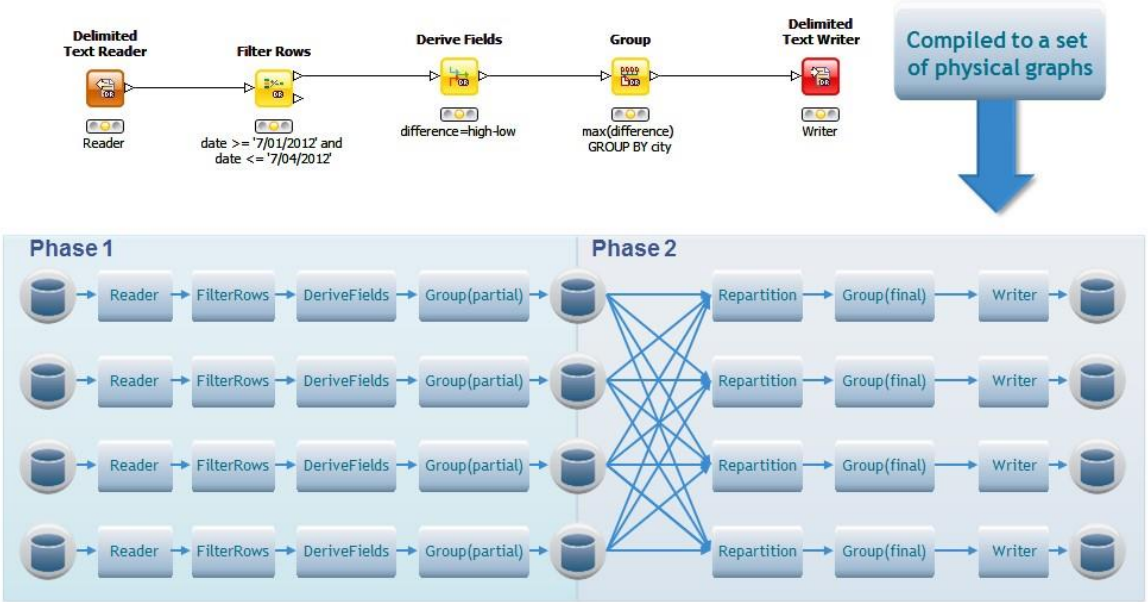
This concept of selecting operators, then stringing them together into workflows is extremely simple for developers, and even non-developers when given an interface, to learn and use.

These operators don't specify HOW a job will be done, but WHAT job the developer wants to do. Below is an example Actian DataFlow workflow.



Each operator has configuration properties which specify details about that particular task. For example, a "Limit Rows" operator will contain a simple Boolean expression that indicates which rows are desired, and which ones should be discarded. A "K-Means" operator would specify how many clusters, maximum number of iterations, the columns to consider, and the type of distance measurement, Euclidean or cosine similarity. A "Missing Values" operator will contain information on what to do if a missing value of a particular data type, or field name is encountered, whether replace with a default value, or discard the row, etc. In this way, extremely complex workflows can be designed very simply.

The DataFlow framework, at runtime, detects how much hardware power is available. It then creates a far more complex version of the original directed acyclic graph, which specifies how each individual operator will be parallelized, as well as optimal parallelization of the workflow as a whole.



Understanding how this works is useful, but since this parallelization is all done by the framework, it's not something that the person developing or executing the workflow has to be concerned with. It is also worth noting that the same type of parallelization will be done, regardless of whether the graph is executed in a single machine or in a cluster.

It should be clear how this style of programming makes development much faster than if the designer had to handle all the tasks of optimally parallelizing workflows manually for each hardware configuration. It also means that a less skilled programmer, or even a nonprogrammer subject matter expert can still create extremely efficient flows that execute at very high speed. There is another aspect of the Actian DataFlow framework that also has a big impact on execution speed.

Pipeline parallelism, data parallelism and in-memory processing

MapReduce and most other parallel programming paradigms are ideal for handling jobs that are "embarrassingly parallel." These jobs essentially involve working on large data sets that are broken into pieces, processing each data set in parallel, then bringing the results back together. This is also known as data parallelism because the same job is done on multiple pieces of data simultaneously.

That is the essence of the map and reduce paradigm. Break the data up, push a processing function (like a data flow operator but generally much simpler) out to that data, read each data chunk off disk, do the processing, push the answer back to a single location, write the answer to disk, then do the next step. This is an extremely efficient paradigm for embarrassingly parallel processing tasks on large data sets in a distributed environment.

Another way to vastly speed up data processing is to read ALL the data into memory first, then do all the processing steps needed on that data, then write the result to disk. This is far faster than the MapReduce concept of read, do one simple function, write to disk, do the next function, but it doesn't scale anywhere near as well. First, there is a high up-front overhead as huge datasets are read all at once from disk to memory, then the limits of RAM in even the most powerful systems can get strained when multiple complex operations are done on a large dataset. When really compute intensive jobs such as full table scans, sorts, fuzzy matching for de-duplication, and complex analytics algorithms are applied, this, otherwise brilliantly fast method, can choke on insufficient resources and die.

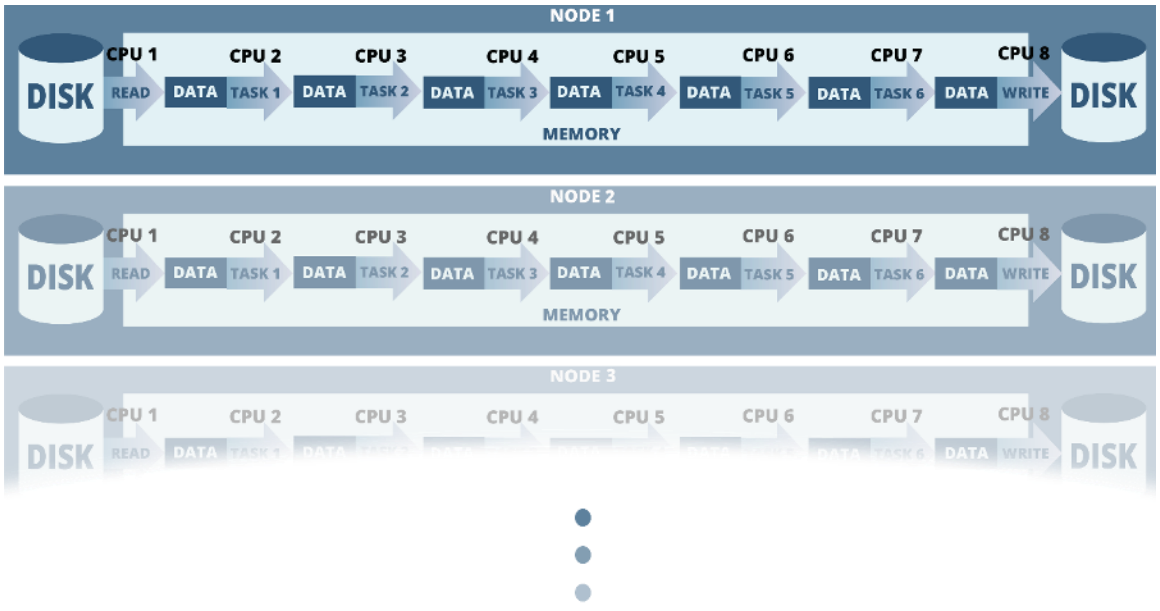
Actian DataFlow does employ data parallelism, just like MapReduce. It does break up data processing tasks and push them out to the various chunks of data in a distributed file system like HDFS, so it gains an equal speed advantage for that type of parallel processing as MapReduce.

However, it also gains much of the speed advantage of in-memory processing without the tendency to overwhelm RAM resources. It does this by employing a second type of parallelism in addition to data parallelism, pipeline parallelism. In pipeline parallelism, multiple functions are executed simultaneously.

To understand pipeline parallelism, it helps to think of the RAM in a server or cluster as the conveyer belt in an assembly line, and the CPU's as workers doing various tasks as the data flows past. Instead of the up-front time cost of reading all data off of the disk into memory, a single chunk, such as a row or column, of data is read, then that chunk is passed to the next function task in the workflow. The second function is being executed on the first data chunk in memory by another thread, while the second data chunk is still being read off the disk. Then when a third chunk is read, the first chunk has already moved on to the third function in the workflow, and the second chunk is on the second function in the workflow, keeping three CPU's working at once. By the time the last chunk of data has been read, most of the data has already been processed and written out to disk.

This also means that only the records currently being processed are held in memory, and only the compute tasks needed on those particular pieces of data are being done, also in memory and simultaneously. When all tasks are done on a particular chunk of data, the answer is written out, and that data discarded from memory, making space for more data and processing. That data, essentially, falls off the end of the assembly line.

Illustration:



If you return to the previous illustration showing the conversion of design graph

to a physical graph, you can see that all of the steps in that workflow were carried out in memory, with the exception of the read, the write and the Group step, which required that data be written to disk and reshuffled. This is a fair representation of most DataFlow workflows.

This combination of both data and pipeline parallelism gives DataFlow performance speed that far exceeds MapReduce, and approaches in-memory, without the in-memory costs or dangers. (If this sounds like Apache Spark to you, see Appendix II for a detailed comparison.)

Analyze ALL the Data

Accuracy is the number one advantage gained by being able to crunch more data in the same amount of time.

Reducing the need for sampling is one way to increase accuracy. Increasing the degrees of freedom, the number of variables an analyst can consider, is another way. Increasing sample size can immediately increase a statistician's level of confidence in data analysis results and can help with more accurately training machine learning algorithms.

"Doing away with sampling would be a massive boon to analytics. The more datasets grow over time, the more sampling is required, and the more radically sampling skews the results."

– *Krishna Roy, 451 Group*

Iterating through and systematically tweaking an analytic model a dozen or even hundreds of times to refine it can be another way to improve accuracy. Fast processing speed makes this much more practical. Refining a model until the statistician is genuinely happy with it can make for a far more confident and accurate model in the end. More on that later, in the section on shortening the whole analytics process.

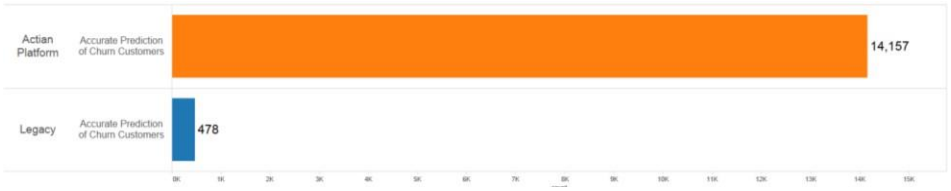
Another big accuracy boost from data processing speed comes when a statistician can include more columns/features/variables into their data analysis. For a greater number of variables to be analyzed with confidence that the results are accurate, a much larger overall volume of data needs to be analyzed. The sample size must be far larger in order to allow more degrees of freedom

without reducing confidence levels in the prediction.

Many data scientists would be delighted to do highly multi-variant analysis, and many of the data sets stored on Hadoop are large enough to support it. Software processing time should no longer hold them back. Some analysts assume that it is the hardware holding them back. In many cases, the "hardware limitations" are really limitations caused by poorly optimized software.

Example: Actian used a machine learning algorithm to predict customers likely to churn soon from a Telecommunications company's call detail record data. With the limited sample size legacy software could process, only 19 columns of the data could be analyzed. Choosing what seemed to be the best indicators, only about 500 potential churn customers were identified. With Actian DataFlow, a far greater sample size could be analyzed in the same amount of time on the same hardware, allowing the data scientist to include 15 more variables. This new data combined with the original 19 variables allowed the algorithm to accurately identify about 14,000 customers likely to churn. The additional variables were very valuable for eliminating possible false positives with surety. At an average cost to a Telecom company of $500 per customer acquisition, the 13,500 churning customers not accurately identified by the previous software could mean a cost to the company of as much as $6.75 million.

(Done on a known test dataset with 16,000 true positives. Dataset available upon request.)



(Variables above the "churn" line on the list to the right were used for legacy analysis, additional variables below churn line were added for the second analysis.)

Enhance Data with Multiple Data Sources

"Actian allows businesses to run their business on data not instinct."

– *Robin Bloor, The Bloor Group*

One of the problems that face enterprises ready to embrace Hadoop as part of their enterprise information management architecture is that Hadoop and MapReduce do not generally work and play well with others. Hadoop users often say, there's no such thing as a "join" in Hadoop, despite the function existing in Hive, due to the limited utility of the function. While an inexpensive way to store and process data is hugely useful in an enterprise architecture, yet another large, isolated data silo is not.

The best possible way to use the data stored in Hadoop would be as if it were simply another data source to feed analytics. In order to do that, a data management tool that CAN efficiently join Hadoop data with other types of data is needed. Once various joins or lookups or other forms of data merging are accomplished, the problem of duplicates has to be elegantly dealt with.

In addition, Hadoop data is notoriously filled with a high percentage of "noise" or non-useful data, and has none of the data quality profiling and cleansing routines applied to it that are so essential to making other types of data useful. Even in Hadoop, "garbage in, garbage out," still applies.

Actian DataFlow addresses these issues in several ways.

Broad and Extensible Connectivity Options

The pre-built Actian DataFlow operators that I mentioned in the previous section provide a wide variety of functionality. Each operator does one specific thing and does it very well in parallel and distributed compute environments. One set of operators are the I/O readers and writers. These, of course, include connectivity to HDFS and HBase, but they also include parallel read and write to a variety of other formats. Virtually any RDBMS can be read and written to with DataFlow through JDBC, as well as text formats such as delimited, log files and sparse data.

Beyond that, DataFlow smoothly integrates with Flume, Sqoop, Storm, Kafka and other Hadoop projects to handle streaming data sources, and, of course, integrates with Actian DataConnect to read and write from nearly every other data source on earth, including on-premise and cloud-based application API's, web services, and hierarchical data like XML and EDI.

And, if that isn't enough, the Actian DataFlow programming framework can be used by any competent Java, Scala, Jython, or other JVM supported language programmer to create new reader and writer components. These can be dropped right into DataFlow workflows right alongside the pre-built operators.

Join heterogeneous data

At a recent Strata conference, a presenter said that on Hadoop, joins were simply not an option. Most of the audience simply nodded in agreement. Hive, Pig and other Hadoop tools that generate MapReduce under the covers, support joins on Hadoop, but they can be difficult to implement, and limited in functionality. MapReduce-based joins, regardless of how they are generated, are notorious for overloading the memory cache on Hadoop clusters. And they are not designed to pull data from other sources outside of Hadoop. At Actian, we do joins on Hadoop all day long. A recent proof of concept for a new project with a long-standing customer involved more than a hundred joins in a single workflow, not counting the internal joins that were done in SQL as the data was pulled from various databases.

Actian DataFlow is not dependent on MapReduce, whether on the surface or under the covers, to manipulate data, so it is free to manipulate non-Hadoop data and Hadoop data at the same time. It is not limited to the Map and Reduce paradigm for data manipulation, and it uses pipeline parallelism, which is more memory efficient in general.

In DataFlow, there are pre-built operators for standard joins, cross joins, semi or anti-joins, and unions. All of the pre-built join operators will join data, regardless of whether or not the data originated from HBase or HDFS, or from any other data source from an enterprise data warehouse to a spreadsheet. It isn't necessary to put all of your data on Hadoop before you can use it.

1. Actian DataFlow has three requirements to do joins:

2. There must be connectivity to the data.

3. There must be a way to identify which records join with which, a key field of some sort that matches up (data must match, but field names do not have to).

There must be a DataFlow reader operator that can feed the data to the join operator.

Configuration for the join operators is pretty straightforward, and completely independent of the source type of the data.

Since the data flow is optimized in a "just in time" manner, there is no need to estimate memory cache size needed, and hope that it isn't exceeded. Exactly the right amount of memory will be allocated at runtime.

This provides the capacity to enhance data with geo codes from lookup tables or verify and standardize addresses with third-party address databases. Data about cumulative website clicks on Hadoop by a user can be combined with that person's transaction history from a data warehouse. The possibilities are endless.

Fuzzy matching to weed out duplicates

Of course, the moment you start merging multiple versions of data from different sources, you have the problem of duplicate data. Actian uses powerful fuzzy matching capabilities to compare, discover, cluster and weed out duplicates. This compute intensive task is another aspect of data analysis that is accelerated by the Actian Dataflow processing speed advantages.

Once the workflow designer indicates the fields to be compared and the comparison weight of each field, a threshold is set for certainty that the records are duplicates. For example, if the math indicates an 85% probability that two records are identical, then one is automatically discarded.

Data Transformation

The next problem faced when combining multiple data sets for enhancement is that data formats are rarely compatible, and never just naturally in the format that is needed for the final desired output.

Actian has a decades long history in ETL and data management (previously Pervasive, and Data Junction before that). Because of that deep understanding of the difficulties inherent in integrating heterogeneous data, a very extensive list of transformation operators has been pre-built into Actian DataFlow. Aggregation, de-duplication, sorting, and joining have been previously mentioned. Operators for filtering, sampling, date manipulation, deriving new fields, normalizing values, data type conversions, field parsing, etc are all built in as well. There is a lot of power in these operators, but they're nothing new. Essentially, the capabilities of any good ETL tool are included in the software.

What is new and particularly useful for big data management needs is that all of these operators are built on the DataFlow framework, and therefore are automatically optimized to run in highly parallel distributed fashion directly on the cluster. Data transformation occurs on the cluster where the data sits. It is not necessary to move the data out of Hadoop, then make the necessary transformations in a single-threaded, aka slow, environment. Nor is this a clunky

MapReduce code generator tacked onto a normally single-threaded ETL tool.

DataFlow is thread-parallel at its base, and this makes it inherently more efficient than process-based parallelism such as MapReduce, even when optimized by a skilled MapReduce coder. It is head and shoulders more efficient than machine-generated MapReduce code.

Data Profiling and Cleansing

The first purpose the DataFlow framework was applied to was the compute-intensive task of profiling data, detecting statistics such as means and ranges, as well as detecting distinct values, missing values, and testing against a variety of business rules. This data assessment is often the first step before corrective cleansing can be implemented. Even on small datasets, it has been known to bog down standard hardware and software systems.

Pre-built operators in Actian DataFlow explore, analyze and summarize data quality at unmatched speed, identifying up front what data cleansing needs to happen. The transformation operators then enable high speed remediation of any issues found.

This is especially useful if you have merged multiple data sets, since you can then assess and remediate the health of the resulting dataset at the same level of speed that it was created.

Analytics on Hadoop

Once all of the various types of data that you intend to analyze have been joined, grouped, sorted, transformed, de-duped, enhanced and cleansed, it's time to do some analytics. Building advanced complex parallel distributed analytics algorithms is no small task. Actian has taken care of a great deal of that for you by, again, pre-building high speed data flow style distributed versions of the most common analytics algorithms.

These work just like the data preparation operators, drag them onto the canvas, double-click, and configure properties. The properties to configure will vary according to the analytics algorithm. For example, the "FP-Growth" operator for market basket analysis asks for the transaction identifier field, the item identifier field, the K value, and the minimum levels of support and confidence.

The list of pre-built operators in Actian DataFlow is fairly extensive and growing rapidly. New operators are added every few weeks by the Actian development team. (See Appendix I for a snapshot of operators.)

Analyze ALL types of data

Context is what you gain when you can combine data from several sources and analyze all the types of data available to you.

In many current enterprise analytic architectures, Hadoop data is filtered, aggregated, and distilled down to a tiny fraction of its original size, removing most of the valuable detail, then that small fraction is analyzed, either in a data warehouse, or an analytics specific database. In some cases, only the Hadoop data is analyzed. In some cases, the Hadoop data and other enterprise data are analyzed separately. In either case, segregating data is never the best way to extract answers and understanding from it.

Joining datasets on Hadoop to get these advantages is not only possible with Actian DataFlow, it's no more difficult than joining any other data sets in an average ETL tool.

Example: Putting a particular customer's website click stream data in the context of their past transactional purchase history will give you a far better understanding of what next best offer advertisement type that customer is likely to respond to than analyzing either one of those two datasets individually. Adding in some geo data might refine the offers enough to improve ad response even further.

Bypass choke points in the analytic cycle

"Data analytics is not an activity. Data analytics is a multi-disciplinary end-to-end process."

– *Robin Bloor, The Bloor Group*

Developing analytics isn't a one and done kind of project. There is a multi-step process involved in bringing an analytics project from the first stages of "What do you want to know?" to "Here is the answer and a recommendation for what action to take."

Different people break up the process in different ways, but we prefer to use the Five Analytic Latencies that Dr. Robin Bloor outlined in the white paper, "Minimizing the Five Latencies in the Analytics Process." These are: Prepare, Develop, Deploy, Execute, Audit.

Actian DataFlow offers technical advantages that shorten the time at each stage in the process, resulting in a cumulative reduction in time to value that is unmatched.

Prepare

Data preparation often eats up as much as 80% of the total time spent on any analytics project. In the earlier section on enhancing data with multiple data sources, there is considerable detail concerning data preparation operators in Actian DataFlow. The advantage of these operators in the area of shortening the whole process is two-fold.

First, the operators are designed to operate in a parallel fashion in a distributed architecture. This means that the execution speed of data preparation tasks is vastly accelerated, compared to traditional ETL applications, and even compared to MapReduce, Pig, or Hive code. (See earlier illustration comparing DataFlow with Pig execution times on standard queries.)

The second advantage is even more significant, because it saves not just machine time, but the more valuable time of humans. Actian DataFlow operators drop into the Eclipse-based KNIME data mining platform. This platform consistently rates highest in the Rexer Analytics surveys on consumer satisfaction for open source data mining platforms, particularly in measures of usability. Using Actian DataFlow operators in the KNIME interface means that data preparation workflows can be created very rapidly, even by non-programmers.

"Actian DataFlow holds a lot of appeal for analytics gurus looking for both design-time and execution-time productivity. The KNIME workflow environment is very accessible and easy to use, and the combined KNIME and Actian DataFlow nodes and operators provide a lot of flexibility with pre-existing building blocks for both ETL and analytics.

The highly parallel Actian execution engine now makes it practical to apply the beauty of the KNIME workflow environment to large datasets that would have been difficult or even impossible to tackle otherwise."

– *Dean Abbott, President, Abbott Analytics*

Anyone with enough understanding of data to operate a standard, GUI-based ETL tool can now do their own cluster data preparation tasks. This saves considerable friction between IT and line of business users and data analysts. If, for the sake of improving an analytics model, an additional data column is suddenly needed, a particular missing value needs a default, negative values need to be set to absolute, etc. the analyst can make that modification in the

workflow themselves, rather than waiting for IT to implement it.

Develop

The advantages of using Actian DataFlow vastly increase when the normal "develop, test and refine" iteration cycle of analytics models is considered. A data analyst no longer spends half their day reading email and waiting for their analysis model to finish executing so they can check it, make a minor tweak, and run it again.

And, with the visual interface, those minor tweaks take a few mouse clicks. That also saves a great deal of the analyst's time.

If the person developing the analytics workflow prefers to have the greater level of control and subtlety of writing code directly, rather than using an interface, the underlying Actian DataFlow framework has a friendly API that will work with any JVM-based language: Java, Scala, Jython, Groovy, whatever you like. It also has a simple JavaScript interface. Regardless of what language you choose, the framework will still handle all the parallelization tasks. You simply need to build a directed graph with the instructions of what you want done, the framework will handle getting it all done optimally, regardless of thread or core availability at time of execution.

Deploy

The deployment step in the process appears to be, anecdotally from the customers and Hadoop users we at Actian have spoken to, the biggest hurdle enterprises face. Production deployment is the choke point that stops many projects flat, and delays others for months, or even years. The reasons for this problem are clear.

First, in many cases, analytics are designed by data analysts. Whether you call them statisticians or data scientists or whatever, they are specialists in the analysis of data. They often build their analytics models in R, Python or SAS, or some other analytics package. They've tested it out using a small sample of data provided to them by their IT departments.

That model is then passed to another person or team. It is this second team's job to modify the model to function in a distributed environment and connect it to all the data needed to actually produce results. This team is generally expert in parallel programming, however, parallelizing a non-parallel analytics workflow and algorithm is not a simple task.

After some weeks or months of programming, the MapReduce developer then tests out the model on the Hadoop cluster and gets back … gibberish.

Something has been lost in the translation. Perhaps the parallel version of the analytics algorithm doesn't work the same as the single-threaded version did. Perhaps the small sample that the model was tested on was not a sufficiently representative sample of the real data.

Regardless of the reason, the end result is that the Hadoop developer and the data analyst have to go back and forth multiple times to try to get the distributed version of the workflow to function in the way that the data analyst expected it to. It can be a very long-drawn-out process, and in the end, it can even be completely unsuccessful. Or, it may simply become a moot point, since business needs move rapidly. By the time the answer is finally available, the business people may want to ask completely different questions.

Actian DataFlow bypasses all of that. As has already been mentioned, Actian's operators are all designed to work in parallel and distributed environments from the beginning. Once the software is installed on the cluster, DataFlow workflows can be developed and tested on a desktop, or directly on the Hadoop cluster itself, by changing a single setting in the user interface. It literally takes seconds. And if you have two Hadoop clusters, one for testing, one for production, the same thing applies. Just change the setting from the test environment to point to the new production environment. Done.

Instead of being the biggest choke point, production deployment with Actian DataFlow is the quickest and easiest part of the process.

Execute

Enough has already been said about Actian DataFlow execution speed. It takes advantage of as much compute power as is available on any hardware platform to squeeze the best possible execution speed out of that hardware.

See the earlier section "Automate execution optimization" for more details.

Audit

Auditing production models can be the biggest key to continuing to get significant lift from those models over time.

"You can't simply deploy a static model. Signals and patterns change. Models must constantly improve, re-learn and update to keep their value."

– *Laks Srinivisan, Opera Solutions*

Having a self-documented workflow in an easy to modify interface is extremely helpful when coming back to a model that the analyst hasn't looked at or thought about in months or even years.

When the machine learning algorithm needs to be re-trained, the workflow tweaked, tested, iterated and tweaked again, then the whole thing needs to be re-deployed, all of the same time savings that were mentioned in the previous sections once again come into play.

ALL the way from end to end

Agility is the main benefit a company gains from a shorter end-to-end analytics cycle.

The time between identifying an analytics need and fulfilling it has to be short enough to actually answer that need. Business needs change so quickly that a 3-month, 6-month, or even 1-year cycle, as is not uncommon in Hadoop implementations, simply isn't practical. The benefits of a shorter end-to-end analytics cycle don't just come into play once. Every time a new analytics need is identified and every time an existing model needs to be updated, this overall time-to-value advantage shows up again. These time savings are cumulative and translate to real dollar benefits.

Example: As a demonstration for a potential client, a KNIME representative built an Actian DataFlow workflow and deployed it on a Hadoop cluster in under 20 minutes. This workflow duplicated one that a team of MapReduce coders had spent more than two weeks coding. The output from the two was nearly identical, except that the predictive results were slightly more accurate from the KNIME job.

Example 2: One of the largest data science consultancy firms in the world chose to use the Actian DataFlow framework directly, rather than through the interface, because they had a wide variety of skilled programmers who were also statisticians in-house. DataFlow improved execution time on a particular risk analysis solution by 90%, reducing their customers' exposure time from 3 days to a few hours. That was enough for them to buy the software. The development time reduction due to the framework automatically handling the parallelization aspects was a nice bonus. But the true benefits to their company appeared when it came time to deploy the solution to many customers, and to deploy other solutions as well that had been built on the same DataFlow framework. Before, using older styles of development and deployment, it took an average of 2-3 days to deploy a new analytics solution for each client. Now, it takes an average of 2 hours. This has allowed them to accept far more clients while still meeting their SLA's, and significantly increased their overall corporate revenue.

Be a good enterprise citizen

"Man plus machine is at the heart of good analytics. Whatever you make must be usable by people in the right context."

– *Laks Srinivisan, Opera Solutions*

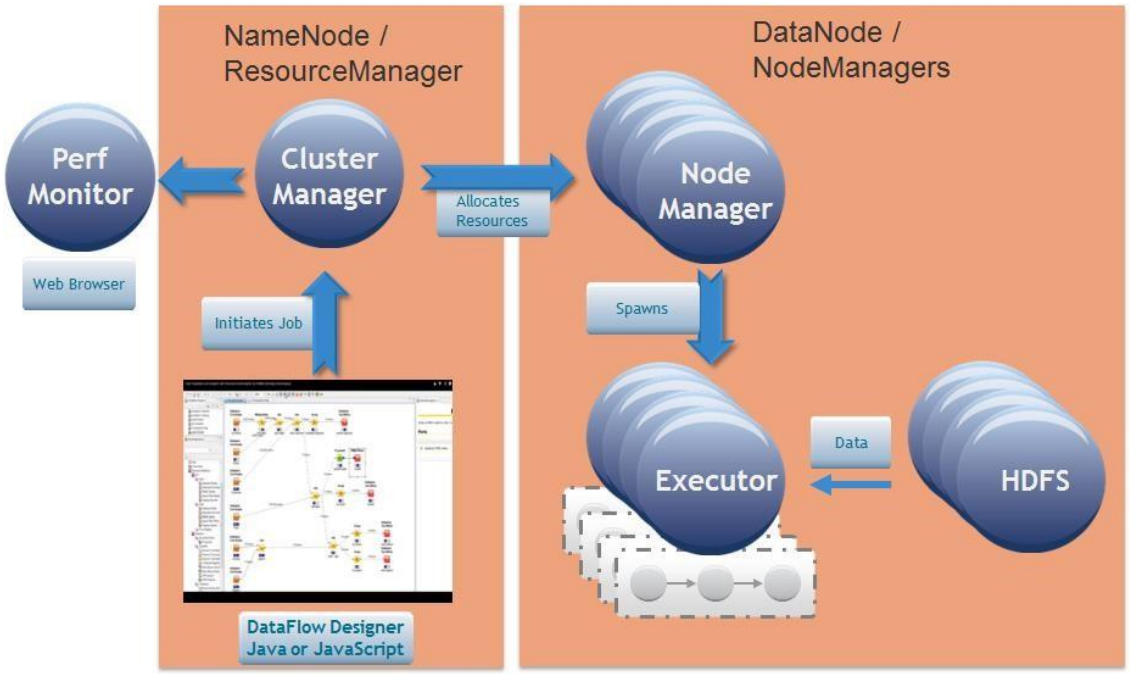
Platform agnostic

Actian DataFlow is software that is platform agnostic. In this case, the word platform applies to hardware, operating systems, and Hadoop distributions.

DataFlow will function optimally on a laptop, a desktop, an industry standard server, a high-performance server, or a cluster of any size made up of nodes of any level of power.

DataFlow runs on any operating system with a JVM, including Windows, Mac, Linux, and various flavors of UNIX.

Dataflow is YARN certified for Hortonworks, and Cloudera certified as well. It will function on any Hadoop distribution: Apache, MapR, IBM BigInsights, etc. DataFlow includes its own cluster management capabilities but will use the YARN cluster resource manager instead with a simple settings change.



Actian Analytics Platform

Actian DataFlow is one component of the Actian Analytics Platform, which also includes Actian Vector, the world's fastest single server analytics database. The real value of the platform shows up when both pieces are combined.

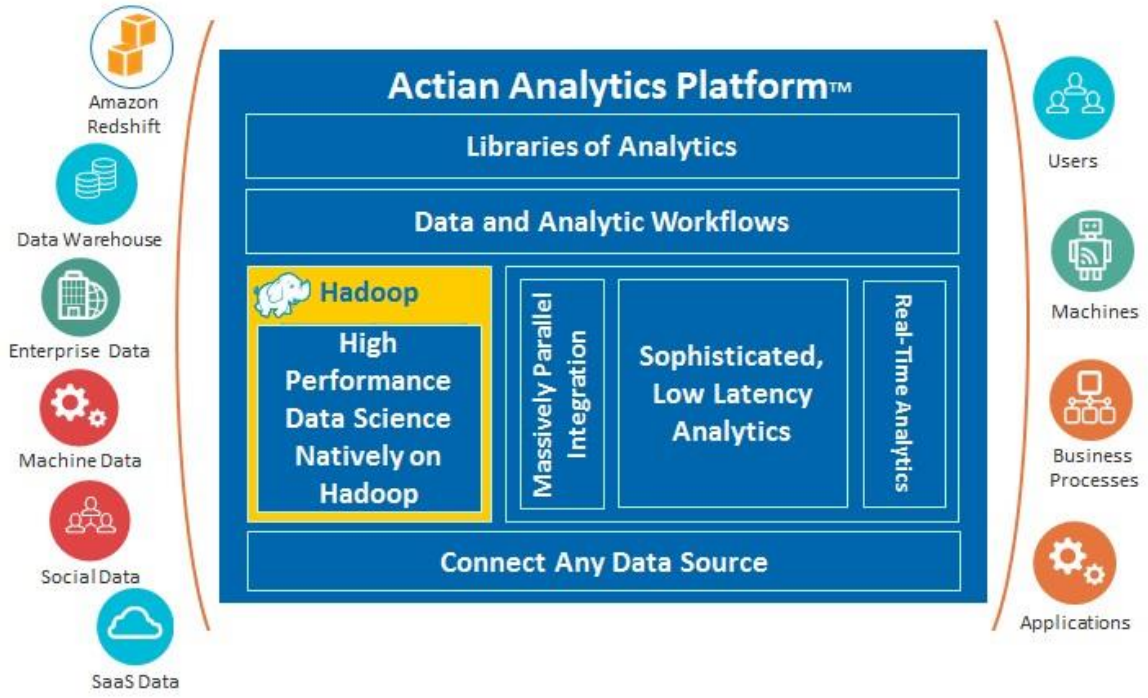
"Opera Solutions is literally betting our business on Actian. We've done that with Dataflow and Vector."

– *Laks Srinivisan, Opera Solutions*

Actian DataFlow can be used stand-alone to do data preparation, high performance ETL and analytics directly on Hadoop. It can also be used for high performance data integration and data quality projects.

Actian DataFlow is ideal for situations where data extraction, preparation, merging and enhancement on Hadoop feeds into the Actian Vector analytic databases.

Hybrid implementations where some scheduled analytics or data mining is done on Hadoop with DataFlow and other, more ad hoc, fast-response analytics are done on Vector are also common.



Enterprise architectures

Actian DataFlow integrates directly with a wide variety of data sources, and through Actian DataConnect to hundreds more. DataFlow can become the backbone of high speed, large data management. It can connect Hadoop with the rest of the enterprise software, making the yellow elephant in the room no longer a big, isolated silo on the edge of the integrated enterprise.

Actian DataFlow is also used in many large scale ETL and data quality projects that don't even involve Hadoop. Being platform agnostic means that DataFlow can run anywhere it's needed.

Other data preparation and analytics tools

One of the distinct advantages of the Actian DataFlow software is that it "works and plays well with others." If there is a particularly brilliant analytics algorithm that brings value to a business that was written in R and doesn't translate well, that R algorithm can be dropped directly into a DataFlow workflow, allowing DataFlow's high-speed data preparation operators to extract, enhance, aggregate and distill the data as needed before feeding it into the algorithm.

Similarly, many enterprise analytics have run on SAS for a lot of years. Those analytics have been refined and refined until re-building them from scratch would be a nightmare. But replacing the SAS ETL with Actian DataFlow can vastly improve performance on those old stand-bys.

Example: One Actian customer took 48 days in SAS to do a fraud detection analysis. 47 ½ of those days were data preparation. Only a few hours were actually spent on the key analytics. Actian DataFlow replaced the data preparation aspect, fed the clean data into SAS analytics, and now that customer catches fraudulent transactions in less than a day, not 48 days.

Actian DataFlow reads and writes PMML to both import and export analytics specifications from other software like SAS, IBM SPSS, Weka, Zementis, etc.

Actian integrates with data visualization software such as Microstrategy, Qlik, Yellowfin, and Tableau. Actuate's BIRT open source version imports right into the same KNIME interface.

Even a complex regular expression or a snippet of Javascript code can be the key a particular project needs. Actian DataFlow lets you drop that right into the workflow where it's needed and will run with it.

Conclusion

"The impossible is now possible. What would you attempt to do if you knew you could not fail?"

– *Laks Srinivisan, Opera Solutions*

DataFlow automatically optimizes execution on any hardware platform, providing the best possible execution speed. DataFlow provides the ability to merge and enhance Hadoop data with other data sources as easily as any ETL tool merges and enhances any other data source, but far faster. It reduces the inherent latencies at all of the steps in a normal analytics process from data preparation through development, testing, refinement, deployment, execution, and updating. DataFlow is completely platform agnostic. It runs on virtually any hardware, operating system or Hadoop distribution. DataFlow is an integral part of the Actian Analytics Platform, and integrates seamlessly with the other essential software in the enterprise. In these ways, Actian DataFlow addresses all of the challenges presented by Hadoop in the enterprise today.

These advantages provide corporations with the ability to analyze **all of the data** for greater accuracy, analyze **all types of data** for greater context, and shortens the analytics cycle **all the way from end to end**, for exceptional business agility.

There are undoubtedly many other advantages not listed in this paper to being able to crunch as much data as desired, as many different types of data as desired, whenever it is desired. Many of those advantages haven't even been thought of yet. Statisticians have been dealing with the limitations of their software's data crunching capabilities for decades, well before the term "big data" became popular. To a large extent, the concept that you can only analyze so much data is built into every aspect of a statistician's daily life.

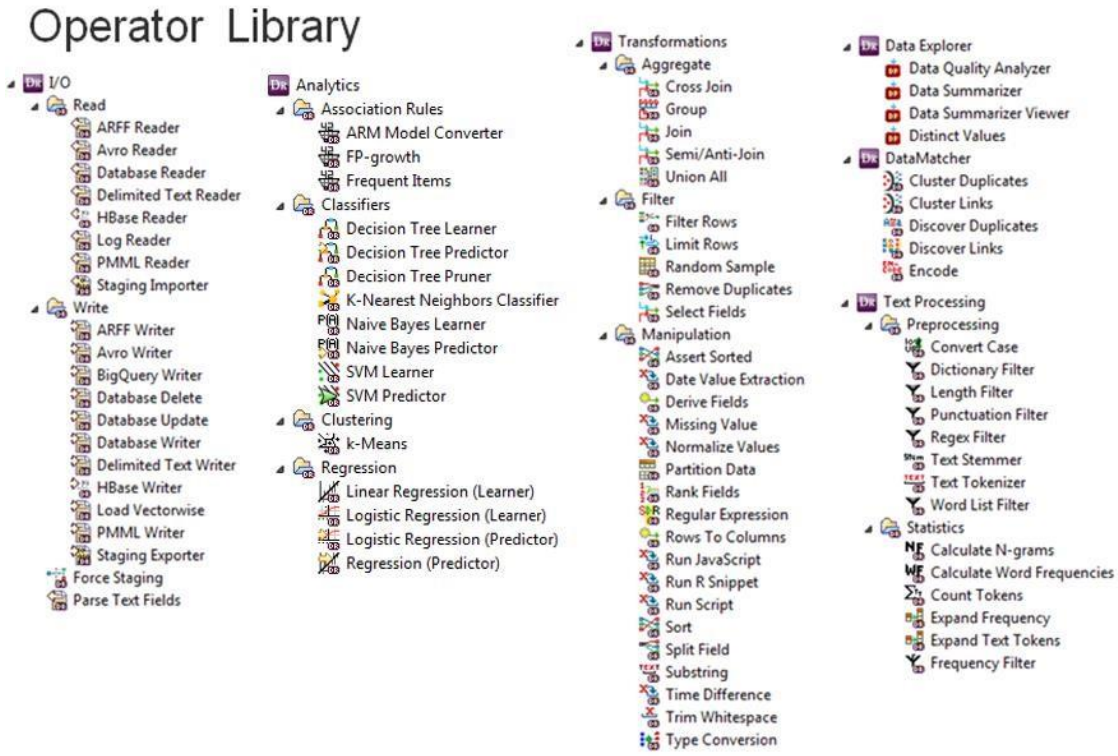
Only when the people who deal with data on a regular basis are set free of those limitations will we truly begin to see what gifted data scientists can really do. This freedom from constraints, and the new possibilities it opens up, is the biggest value that Actian DataFlow offers. This benefit can't be concretely counted, but Actian believes that in time, this new freedom will revolutionize data analysis.

Appendix I

Operator List

The list of pre-built operators in Actian DataFlow is fairly extensive and growing rapidly. New operators are added every few weeks by the Actian development team.

The operator snapshot illustration below should give you a general idea of the pre-built operators available in the user interface version with KNIME. If you don't see a specific piece of functionality, don't assume that it doesn't exist in Actian DataFlow. Some DataFlow capabilities are not in the user interface, and by the time you read this, the list will be incomplete.



Appendix II

Comparing DataFlow and Spark (and Tez, Pig, Hive, Mahout and Cascading)

Apache Spark is an open source project originally developed in the AMPLab at UC Berkeley to improve data processing speed on Hadoop. This section will compare and contrast the two technologies.

Spark, like Actian DataFlow, uses a directed acyclic graph execution engine that does just in time workflow parallelization and optimization. Also, like DataFlow, Spark is not tied to the inflexible map and reduce programming paradigm. The two frameworks have, in many ways, very similar strategies for speeding up processing of data on Hadoop clusters.

Both Spark and DataFlow do not use MapReduce in any way. Both will coexist peacefully with MapReduce programs (and each other) on the same cluster and share resources using the YARN resource manager. Both will vastly speed data processing over any strategy that does rely on MapReduce under the covers, such as Hive, Pig, Mahout or Cascading, since they are not tied to the highly restrictive Map, Sort and Shuffle, Reduce paradigm. Both DataFlow and Spark are efficient ways to do high speed data processing and analytics on clusters.

Aside from DataFlow being proprietary software, and Spark being open source, DataFlow and Spark have slightly different strategies for attaining that high execution speed. DataFlow, as explained earlier in this paper, uses pipeline parallelism as its main strategy for high speed execution. Spark uses an entirely in-memory processing paradigm.

Spark starts by reading all data into RAM in a data store called a Resilient Distributed Dataset (RDD). This in-memory dataset can then be queried or read repeatedly, which is extremely useful, particularly for certain machine learning algorithms that require iteratively reading over a dataset multiple times.

The RDD concept seeks to solve the traditional in-memory problem of fault tolerance. Normally, if the machine holding an in-memory data store fails, all information is lost, since the RAM is lost. Spark writes a lineage model to disk that tracks where the data came from, and all processing steps that have been done to it. In this way, the RDD can be re-generated in case of a fault. So, unlike most in-memory processing strategies, Spark is reasonably fault tolerant.

Tez is another project that uses an in-memory strategy and skips many of the limiting programming restrictions of MapReduce to speed up processing.

However, it does not have the concept of the RDD. Therefore, it is not fault tolerant, and it cannot do the high-speed iterative processing that makes Spark so advantageous, particularly for machine learning and predictive algorithms.

One difference between Spark and DataFlow is in the way they do sorting, a highly compute intensive operation, and often the slowest point in any workflow. When a sort is done on data, DataFlow records to disk the new dataset with the data sorted, and it will remain sorted for the rest of the workflow and beyond. Spark and DataFlow can do roughly the same level of high speed sorting, but Spark does not retain the information in the new sorted form, so it may be necessary to sort data repeatedly.

Another difference is the lack of pipeline parallelism in Spark. Because of this, only one function can be executed in Spark at a time. That function must be completed over the entire dataset before the next function can begin. Pipeline parallelism allows many functions to execute simultaneously. By the time all data has been read into memory, and Spark can start processing it, DataFlow has already completed processing most of the data.

The main disadvantage of Spark is that it still has the limitation of any in-memory processing paradigm in that it bumps against limits in RAM availability. The entire data set is read into RAM in the form of an RDD first. When a data processing step is done that alters the data, a new RDD is created. Having multiple copies of the RDD in memory can become particularly RAM intensive, especially if many alteration steps are needed. Since only small amounts of data are in the DataFlow memory pipeline at a time, DataFlow does not tend to overwhelm RAM resources. Spark is smarter than many in-memory paradigms in that it can spill some of its processing to disk when RAM limits are reached, and therefore not cause a crash or freeze, or even a processing failure, but in that case, much of the processing speed advantage is lost.

Execution speed in general tends to be roughly equal between Spark and DataFlow. Depending on what particular task is being done, Spark may be faster, or DataFlow may be faster. An example of a workflow where DataFlow would tend to execute more quickly would be one that required sorted data to be accessed multiple times, or one that had a lot of data processing steps. An example of a workflow where Spark might execute more quickly would be one where the full dataset needed to be read multiple times.

Another disadvantage of Spark is that, like MapReduce, Spark requires very specialized skills to use. It is essentially a Scala programming framework. Python and Java API's are available, but these API's tend to trail behind the main Scala API. In addition, not many pre-built operators for Spark exist at this time (although that is changing). This means that Spark requires another kind of specialized skillset, a skilled parallel Scala programmer.

Dataflow can be used by any decent Java programmer, even with no parallel programming skills, or by anyone who can understand the data and processing steps needed well enough to drag, drop and configure pre-made operators in an interface. In general, the ease of developing with Actian DataFlow, particularly with the drag and drop interface, provides a much shorter development time than Spark, minutes as opposed to weeks or months.

When choosing a Hadoop software platform, be sure to consider not just runtime execution speed, but the particular purpose you are likely to put that platform to, the resources required, both human and hardware, and the long-term efficiency of the entire end-to-end analytics process.

About Actian – Activate your Data™

Actian, the hybrid data management, analytics and integration company, delivers data as a competitive advantage to thousands of customers worldwide. Through the deployment of innovative hybrid data technologies and solutions Actian ensures that business critical systems can transact and integrate at their very best – on premise, in the cloud or both. Thousands of forward-thinking organizations around the globe trust Actian to help them solve the toughest data challenges to transform how they run their businesses, today and in the future. For more, visit https://www.actian.com.

2300 Geng Rd, Suite 150, Palo Alto, CA 94303
+1 888 446 4737 [Toll Free] | +1 650 587 5500