www.ijecs.in

International Journal Of Engineering And Computer Science ISSN:2319-7242

Volume 4 Issue 4 April 2015, Page No. 11558-11562

In-Place Transformation Engine for Analytics

Nilendra Chaudhari, Bipin Patwardhan

Technical Architect
Research & Innovation, IGATE Global Solutions, Mumbai, India
nilendra.chaudhari@igate.com

Senior Technical Architect
Research & Innovation, IGATE Global Solutions, Mumbai, India
bipin.patwardhan@igate.com

Abstract: ***Enterprises collect huge volume of data from different sources such as web logs, click stream, social media, sensor data and the like. This includes numeric and textual data and needs to be converted onto a format (mostly numeric) that is acceptable by the data mining and machine learning algorithms. Thus, we need an efficient, scalable approach to enable not only simple conversion, but also transformations like normalization. In this paper, we present the IGATE In-Place Transformation Engine (IPTE), which can be used for conversion and transformation operations. A comparative evaluation of commercially available ETL tools and the need for a custom-tool (IPTE) is justified. Three distinct flavours of IPTE implementation are described -- (a) Stand-alone using Java Multi-threading (b) Distributed using Hadoop API's, and (c) In-memory using Apache Spark. By comparing the performance of the transformation process on different data sizes using these three flavours, specific recommendations are made when each of these should be used. Some use cases where IPTE has been used effectively are also presented.***

Keywords: Transformation, Data Mining, Data Preprocessing, Hadoop, Machine Learning, Data Science, In-Memory Computing, Multi-Threading

Businesses today are generating huge volume of data from different sources like web logs, click stream, social media, sensor data, etc. For example, before the common use of smart meters, a meter reading was available once a month. Now, smart meters are generating readings every 15 minutes or so for every customer. Thus, a utility can expect a few million records every 15 minutes, dramatically increasing the computing power needed to process data. Getting insights from such data is of prime importance for enterprises to remain competitive. To generate insights, enterprises have started applying various data mining and data science techniques. As data is received is varied formats, it needs to be cleaned before further processing. Additionally, since most data science techniques work on numeric data sets, the cleaned data needs to be converted into a suitable format. In many cases, simple conversion does not suffice and the data has to be transformed. After exploring some of the tools in the market, we have developed a high-performance, scalable solution, known as the In-Place Transformation Engine (IPTE). The rest of the paper presents more details about the solution, followed by an analysis of its performance. Before covering the solution, we cover the well-known concept of Extract-Transform-Load (ETL).

1.1 What t is ETL?

ETL is a process typically used in Data Warehouse operations, for fetching the data from multiple heterogeneous/homogenous sources, performing various transformations like cleansing, formatting, aggregation etc. and uploading the results into the Data Warehouse.

1.2 Challenges with ETL

Given the maturity of the technology, there are a host of tools in this space, offering varied and rich support. But, many ETL tools are not able to cope with voluminous data. While many tools support Hadoop [1, 3, 4, 7], most operate as a tight coupling of a trio of operations, namely Extract, Transform and Load. It is difficult to separate functionality into individual pieces [8]. In the sections below, we cover some of the issues in using ETL tools, when considering the newer data challenges, one of them being huge volume.

1) Increasing data volume

Even though many ETL tools have updated their techniques to improve the performance, it is becoming increasingly difficult and costly to transform data, to account for the increase in volume.

2) Cascading Errors and painful recoveries

Poorly managed dependencies can lead to process errors that cause jobs to fail. Such errors may not stop at the level of the job that has failed. They can cascade through the workflow and propagate to multiple downstream systems.

3) Transformation Layer tightly coupled with

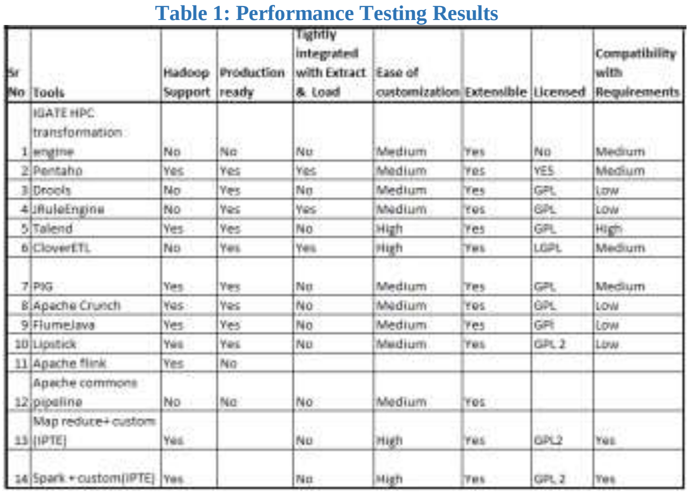
Extract and Load

One of the major problems with the ETL setup is that data is never local and needs to be transferred over the wire (Extract) before it can be transformed. This transfer severely affects performance as the network becomes the weakest link in the setup. Post transformation, data needs to be loaded into the target system, which also involves the network.

2.1 Why a Custom Engine?

For an analytical solution at IGATE, we needed a scalable and reliable method to convert data. Obviously, manual conversion was out of the question, given the volume of data. Over the years, many solutions, including ETL tools, rule engines, etc have been used to convert data. But with the ever increasing volume of data, we decided to evaluate many tools, for various parameters like feature-set, ease of customization, license, and cost of the tool and most importantly, support for Big Data technologies. For this purpose, we evaluated 14 tools – Pentaho, Drools, Talend being a few of them as shown in Table 1 below. We noted that many of the tools were not well suited for integration with Big Data technologies. Thus, for huge volume of data, the conversion tasks could run for many hours. While some tools have Big Data integration – thus being able to scale the transformation portion – their 'extract' and 'load' processes present a bottleneck, due to the fact that data has to be extracted from the source system before transformation and it is loaded into the target system post the transformation. Due to licensing cost, we had to set aside a few tools like Pentaho, while other tools like Drools [10] and JRule Engine [12] were rejected due to lack of support for Big Data technologies. While Talend Studio [11] was found to be fit for purpose, it was set aside for later consideration as it has support only for Apache Hadoop 1 and does not support Apache Hadoop 2 architecture. After a detailed comparison of various tools, we decided to implement a custom transformation engine.

Table 1: Performance Testing Results



Sr No	Tools	Hadoop Support	Production ready	Tightly integrated with Extract & Load	Ease of customization	Extensible	Licensed	Compatibility with Requirements
1	IGATE HPC transformation engine	No	No	No	Medium	Yes	No	Medium
2	Pentaho	Yes	Yes	Yes	Medium	Yes	YES	Medium
3	Drools	No	Yes	No	Medium	Yes	GPL	Low
4	JRuleEngine	No	Yes	Yes	Medium	Yes	GPL	Low
5	Talend	Yes	Yes	No	High	Yes	GPL	High
6	CloverETL	No	Yes	Yes	High	Yes	LGPL	Medium
7	PIG	Yes	Yes	No	Medium	Yes	GPL	Medium
8	Apache Crunch	Yes	Yes	No	Medium	Yes	GPL	Low
9	FlumeJava	Yes	Yes	No	Medium	Yes	GPl	Low
10	Lipstick	Yes	Yes	No	Medium	Yes	GPL 2	Low
11	Apache flink	Yes	No					
12	Apache commons pipeline	No	No	No	Medium	Yes		
13	Map reduce+ custom (IPTE)	Yes		No	High	Yes	GPL2	Yes
14	Spark + custom(IPTE)	Yes		No	High	Yes	GPL 2	Yes

2.2 What is IPTE?

The IGATE In-Place Transformation Engine (IPTE) is a custom rule-based Java engine that transforms data, using well-defined rules. IPTE also supports Hadoop engine and is thus scalable to handle huge volume of data [9]. The biggest differentiating feature of the engine is that it focuses only on one feature, namely transformation. By decoupling the extract and load portions, IPTE is able to apply the power of Hadoop, to the task of transformation.

IPTE is built using the 'pipeline' design pattern, with the rules being specified using an XML syntax. The rules used in IPTE are simple, POJO classes, that are loaded and executed as per the configuration file. It is important to note that IPTE applies all the rules, as defined in the configuration file, one after the other on one of input data. By doing so, the number of I/O operations needed is equal to the number of lines in the input. Additionally, as each line is transformed individually, IPTE can achieve high-performance by taking advantage of Hadoop's massive parallel execute feature. While we have developed some rules like 'replace character', 'skip blank line', 'skip column', the library of rules is expected to grow as per user need

IPTE has been developed with three execution flavours in mind, namely 'Standalone', 'Distributed' and 'In-Memory'. The 'Standalone' flavour uses multi-threading features of Java, to achieve parallel execution, which creates a collection of threads and hands of each line of input data to one thread. In the 'Distributed' flavour, data is stored on HDFS and rule execution uses the Hadoop API for transformation. The 'In-Memory' flavour uses Apache Spark [2], to use the speed of Random Access Memory (RAM), to store the data as well as perform the transformation.

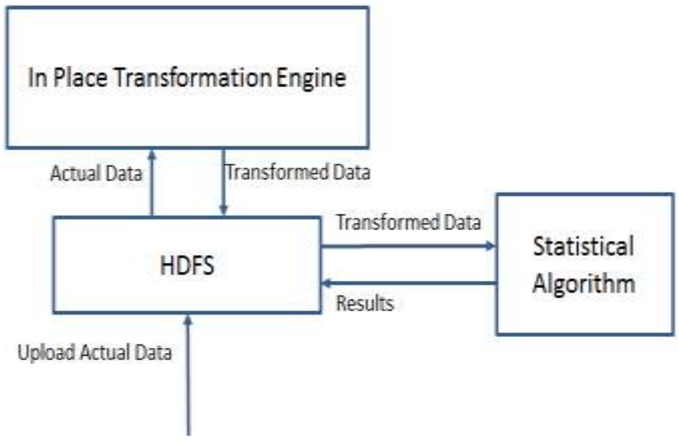


Figure 1: Data Flow

Error! Reference source not found. shows the flow of data in IPTE's 'Distributed' flavor. Data to be transformed is loaded to HDFS. IPTE selects the data row by row and applies the transformation rules defined. The transformed data is stored back to HDFS. Downstream applications can either access the transformed directly, or separate, independent load processes can be used to upload the data to the downstream system's data store.

2.3 Architecture

Error! Reference source not found. shows the high-level architecture of IPTE. IPTE loads rules defined in XML. Data is read line by line from HDFS and passed to the pipeline, where the defined rules are applied. After transformation, the data is written back to HDFS.

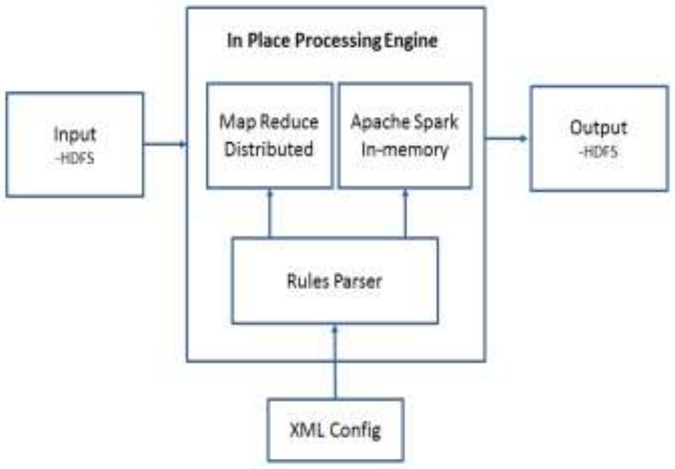


Figure 2: High-level Architecture

2.4 Technologies

IPTE is a Java-based, custom developed component that uses the 'Pipeline' pattern. In the 'Standalone' flavour, it used Java multi-threading; Hadoop API in the 'Distributed' flavour; and Apache Spark in the 'In-Memory' flavor.

1) **Apache Hadoop:** Hadoop is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than relying on hardware to deliver high-availability, the library is designed to detect and handle failures at the application layer, thus delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures [1].

Data is stored in distributed manner using the Hadoop Distributed File System (HDFS). Distributed processing uses the Map-Reduce model (Hadoop 1) or YARN (Hadoop 2).

2) **Apache Spark:** Apache Spark is a fast and general cluster computing system for Big Data. It uses in-memory primitive to improve performance and can use different storage, HDFS being one of them.

2.5 Features

The features of IPTE are

- It operates in three flavors, namely Standalone, Distributed (on top of Hadoop) and In-Memory using Apache Spark.
- Supports Xml based configuration for the rule pipeline.
- New rules can be added easily.
- Completely decoupled from extract and load operations, for high performance

2.6 Benefits

The benefits of IPTE are

- Handles huge volumes of data
- Delivers high-performance as it supports distributed processing and caching mechanisms
- Enables easy-to-define pipeline of execution
- Provides easy mechanism to add new custom rules
- Allows ready integration of standard Java libraries

In the following paragraphs, we have analyzed IPTE's performance in detail.

3.1 Test Setup

The following hardware and software configuration was used for performance testing

- CPU - Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
- No of Cores - 4
- RAM - 4 GB
- OS - Red Hat 4.4.6-3
- Software: Java 1.7, Apache Hadoop 2.2.0, Apache Spark 1.3.0
- Data: Dataset varying from 10MB to 6GB.

3.2 Test Scenarios

IPTE was testing using the following scenarios

1) **Standalone**: IPTE read and write data from local file system. Data and transformation is performed on one system.

2) **Hadoop on single node**: IPTE uses Map-Reduce for computation and data is stored on HDFS. Both, the master node and the slave node are on same system.

3) **3-node Hadoop cluster**: ITPE uses Map-Reduce on the cluster for distributed processing. One system is master node and two systems are slave nodes.

4) **In-Memory**: IPTE uses In-memory for computation and data is stored on HDFS. Both, the master node and the slave node are on same system.

3.3 Test Results

The test results are presented in 2 and Figure 3. The size of the data set ranges from 10MB to 5.9GB (12,500 to 73.75 million records).

Table 2: Performance Testing Results

	Single Machine (Time in ms)			3 Nodes (Time in ms)
Size (MB)	**Standalone**	**Hadoop**	**In-Memory (Spark)**	**Distributed Hadoop**
10	2873	16865	6348	42342
49	12624	19174	16693	54424
97	24782	25365	28990	55381
175	45771	33199	51547	64881
350	87442	62528	99521	70023
700	179948	107004	191522	108825
1000	248808	142155	283582	146437
2000	505166	256786	571824	191847
5900	1526606	744792	1952256	359203

3.4 Observations

Based on the performance results, we have the following observations

- For small files, time taken by Hadoop is more. This is due to the time taken to initialize the Hadoop eco-system and minimum block size allocation.
- For small data sets, the 'Standalone' flavour

performs better than the ‘Distrbuted’ flavour.

- During transformation, Hadoop uses all the cores available on commodity nodes.
- Time taken by ‘Standalone’ flavor and ‘In-Memory’ flavor increases exponentially, as the data set volume increases.
- On commodity hardware, the ‘In-Memory’ flavor takes more time as it is limited by the amount of RAM available for processing – 4GB – which it has to share with the operating system and other processes. For voluminous data, Apache Spark uses virtual memory (hard-disk space), which impacts performance.
- For optimum performance, a server with large RAM is a mandatory. The size of RAM has to be determined by the volume of the data to be transformed.
- The three node Hadoop cluster processes around 12.5 million records or 1GB of data per minute on commodity systems.
- IPTE performance scales horizontally, to execute on available Hadoop nodes.

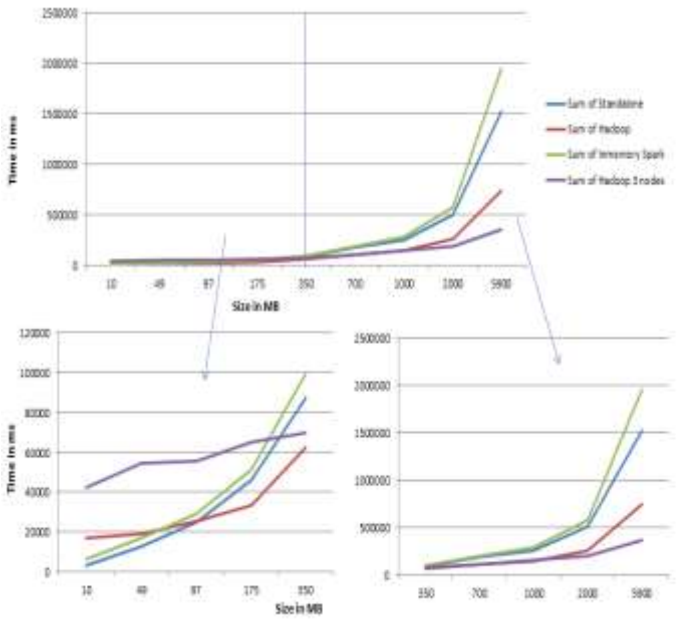


Figure 3: IPTE Performance

Based on the test results, we also have the following recommendations regarding the use of IPTE

1. Use ‘Standalone’ flavour for small data sets – say a million records.

2. For huge volume of data, use the ‘Distributed’ flavour, as it can scale to any volume, simply by adding new nodes to the cluster.

3. The ‘In-Memory’ flavour should be used when extremely fast response is needed. Deploying this solution on a system with insufficient RAM [13, 14], will result in degraded performance.

4. ‘Distributed’ flavour should not be used for small data sets [15, 16, 17].

In this section, we present some of the use cases where we have implemented IPTE.

1) Customer segmentation using K-Means

Customer segmentation [5] is the technique of dividing a customer base into groups of individuals that are similar, based on attributes such as age, gender, interests, spending habits and so on. Typically, customer data is mixed format with some attributes being numeric (like age) and some being non-numeric (like gender). K-Means is an algorithm that uses numeric data as input. Hence, we defined a ‘customer transformation’ pipeline using IPTE to convert customer data into numeric data that can be used as the basis for clustering. One of the rules was to map gender values to 0 (male) and 1 (female).

2) Attribute based Recommendation

Recommender Systems (RSs) are software tools and techniques that can be used to provide suggestions for items. Suggestions can relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read. Attribute based recommendations uses user attributes / item attributes for recommendation [6]. Similar to customer segmentation, custom attribute information and item attribute information (used by the recommendation system) was transformed by IPTE into corresponding numeric equivalents.

3) Data Masking:

Data Masking is a technique where sensitive personal or business data is hidden from unauthorized persons. In most cases, sensitive information is replaced by real-looking, innocuous data. This method of data transformation is commonly used in testing. Presently, we are implementing IPTE in a data masking solution for Big Data testing. In the solution, the source data is stored on HDFS, but not available to the testers. Only selected portions of the data are made available for testing, where the data selection is performed using an IPTE transformation pipeline Given that we are dealing with a Big Data application, we need to test the scalability and volume processing capabilities of the application (that needed to be tested). By using IPTE, it is possible to define multiple transformation pipelines such that the source data gets transformed into multiple sets, depending on the test conditions to be executed. By using IPTE, it is possible to keep the data safe, by allowing only a few select users and IPTE to access it.

As discussed in this paper, we started with the need to have a scalable, high-performance data transformation engine that can be used as part of the analytics solutions that IGATE develops for its customers. After surveying multiple tools, we decided to go for custom development of the transformation engine, where-in we decided to decouple the high-performance transformation portion from the extract and load operations.

To account for varying needs of projects, IPTE has been implemented in multiple flavours. Customers can choose a suitable implementation, as per their performance needs. While presently the rule library supported by IPTE is a bit limited, new rules can easily be added, to enhance the power of IPTE. Implemented, in-progress as well as new use cases are expected to help grow the rule library.

References

[1] Apache Hadoop, https://hadoop.apache.org/

[2] Apache Spark, https://spark.apache.org/

[3] Mike Ferguson "Offloading and Accelerating Data Warehouse ETL Processing Using Hadoop", Seventh Intelligent Business Strategies, 2013. http://info.mapr.com/WP_OffloadingandAcceleratingDataWarehouseETLProcessingUsingHadoop.html

[4] Stephen Swoyer "Using Hadoop to Turbocharge ETL the Smart Way", TDWI, 2013.

[5] Sanghamitra Mitra, Nilendra Chaudhari "Scalable Approach for Analytics based Customer Segmentation", CSI Communications, vol. 38, no. 4, July 2014.

[6] Sanghamitra Mitra, Nilendra Chaudhari, Bipin Patwardhan "Leveraging Hybrid Recommendation System in Insurance Domain", International Journal of Engineering and Computer Science, Oct 2014.

[7] Making the Elephant Dance, http://gcitsolutions.com/?p=2149

[8] Parallel ETL Tools Are Dead, http://blog.syncsort.com/2012/08/parallel-etl-tools-are-dead/

[9] Adopting Hadoop In the Enterprise, http://www.induscorp.com/sites/indus/files/uploads/adoptinghadoopintheenterprise.pdf

[10] Drools, http://www.drools.org/

[11] Talend, https://www.talend.com/

[12] JRulesEngine, http://jruleengine.sourceforge.net/

[13] Apache Spark Hardware Provisioning, https://spark.apache.org/docs/latest/hardware-provisioning.html

[14] Putting Spark to Use: Fast In-Memory Computing for Your Big Data Applications | http://blog.cloudera.com/blog/2013/11/putting-spark-to-use-fast-in-memory-computing-for-your-big-data-applications/

[15] Apache Hadoop: Best Practices and Anti-Patterns | https://developer.yahoo.com/blogs/hadoop/apache-hadoop-best-practices-anti-patterns-465.html

[16] The Small Files Problem | http://blog.cloudera.com/blog/2009/02/the-small-files-problem/

12 key steps to keep your Hadoop Cluster running strong and performing optimum | https://cloudcelebrity.wordpress.com/2013/08/14/12-key-steps-to-keep-your-hadoop-cluster-running-strong-and-performing-optimum/