Associates

ELTMaestro for Spark: Data integration on clusters

Introduction

Spark represents an important milestone in the effort to make computing on clusters practical and generally available.

Hadoop / MapReduce, introduced the early 2000s, allows clusters to be built from commodity hardware by implementing fault tolerance of the cluster's nodes – so a multi-node system could survive the crash of an individual node. But the MapReduce paradigm involves many iterations of high-volume dataflows between memory and disk, which impact performance and make MapReduce unsuited for many of the kinds of computations that data scientists want to use clusters for.

More recently, Spark has received a great deal of attention, and has in many cases become a de facto replacement for MapReduce, as it addresses many of MapReduce's shortcomings. Spark can perform more of its computation in memory without landing data to disk, which in many cases gives Spark order-of-magnitude performance advantages over MapReduce. Spark is written in Scala and leverages the Scala functional programming model to implement fault tolerance. Scala also provides Spark with a convenient and popular API.

Spark contains rich structures called *dataframes* which are used to represent relational tables. Dataframes are *distributed,* meaning that parts of them live on each node in the cluster. The result is that relational operations on dataframes, such as joins, filters, and aggregations take place in parallel, and are very efficient. This makes Spark clusters potentially very useful as data warehousing and business intelligence platforms, since the ability to do such operations on large tables quickly is precisely what DW and BI platforms require.

Spark clusters also have a distinct *dis*advantage as a DW/BI platform – namely, that unlike traditional DW/BI platforms like DataStage or Informatica, there is a dearth of tools for loading relational data into them, and for transforming and organizing that data into the forms required by DW/BI applications. Spark is rapidly evolving, and it is not always easy for developers to keep up with the latest practices. In fact, important changes to Spark have occurred while this paper was being written. Spark has the potential to change the way data warehousing is done, making large scale warehouses more ubiquitous and easier to build -- but its attractive aspects of efficiency and low cost are surrounded by a thorny wall of inaccessibility.

ELTMaestro for Spark is our solution to address this gap – the sword that cuts through the wall of thorns. ELTMaestro for Spark is a new data integration tool that allows users to load, transform, and organize data on Spark clusters using an intuitive visual dataflow language that is familiar to users of traditional ETL tools such as Informatica or DataStage. Unlike traditional ETL tools, however, ELTMaestro for Spark is powered by the Spark cluster itself and has no independent transformation engine. This fact has important consequences in terms of the *cost*, *complexity*, and *performance* of the system – simply put, ELTMaestro for Spark is superior to traditional ETL in all three dimensions.

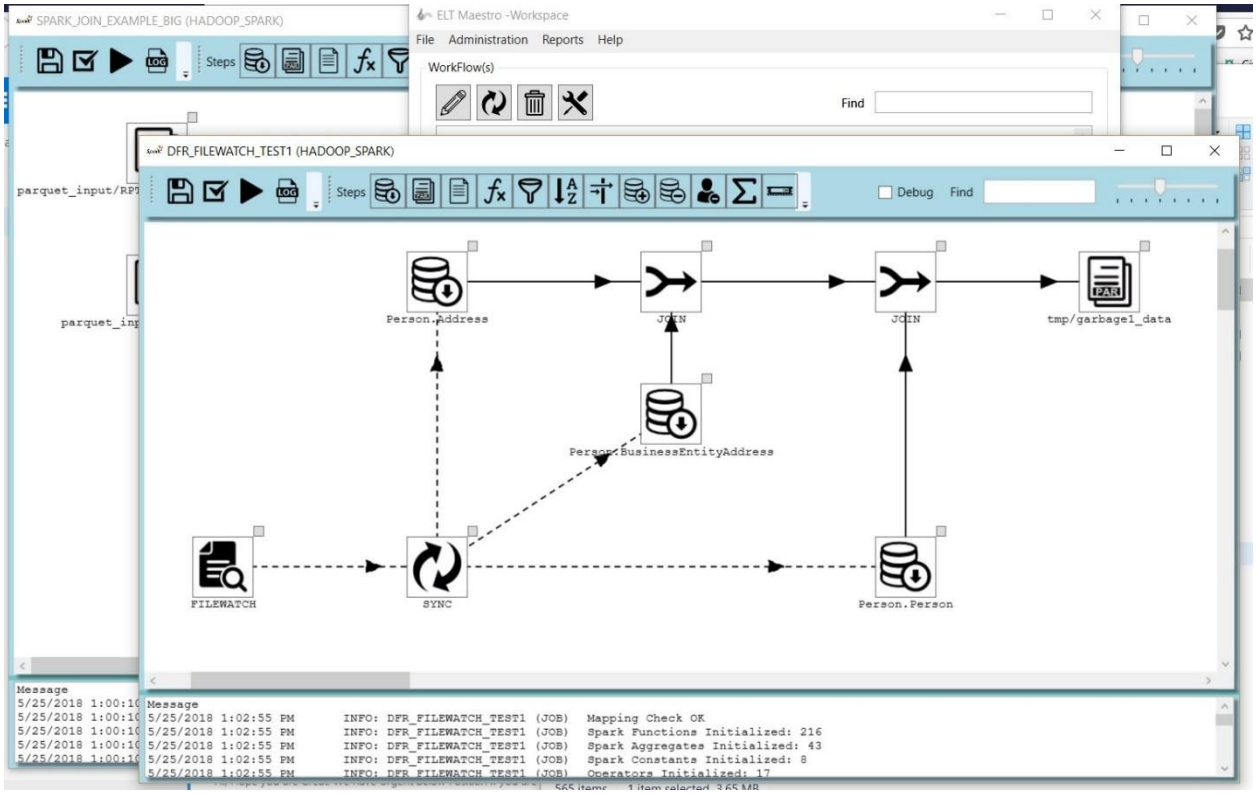

Figure 1

In the next few sections we'll explain some of the principles underlying ELTMaestro for Spark, place ELTMaestro for Spark in the context of other data integration tools and introduce some of ETLMaestro for Spark's advanced features.

ELTMaestro for Spark: An ELT solution

ELTMaestro for Spark is based on an architectural concept called *ELT.* To understand the advantages of ELTMaestro for Spark over other data integration tools, it is necessary to understand the difference between ETL and ELT and the reasons why ELT, when feasible, is preferred.

A brief history of ETL

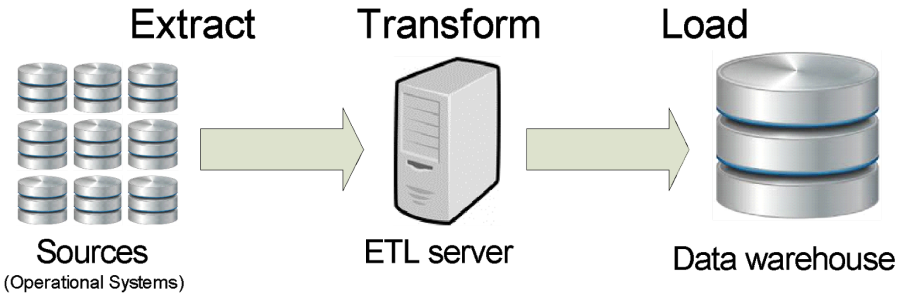

Figure 2

When data engineers first began to develop data warehouses, they basically architected them according to the diagram in Figure 2. Figure 2 is essentially a physical embodiment of how the data engineers thought about the problem – "You have to ***e****xtract* that data from the operational systems (i.e., the sources), rearrange (that is, ***t****ransform*) the data into the form needed for access by analysts, and then ***l****oad* it in that form to a location where it can be accessed."

There were (and are) well-developed schools of thought –notably, those of Ralph Kimball and Bill Inmon – about how the data should be organized so that analysts using the warehouse would be able to access what they needed most easily, and so that the warehouse would grow in a manageable way over time. It was generally thought that the success of the data warehouse depended on getting this organization right.

In this environment, the ETL Server was considered a critical component – the ETL server, after all, is the component responsible for the organization of the data warehouse. With data volumes inexorably increasing, ETL servers, including the software that ran on them, grew ever more complex and expensive. Today, ETL servers for large DWs running IBM DataStage, Informatica, or Ab Initio can be expected to cost on the order of a million dollars.

Meanwhile, trouble developed on the right side of Figure 2. Despite all the attention paid to organizing the data in the DW according the theories of Kimball or Inmon, data in the DW was found to to be increasingly inaccessible. The problem was not so much one of warehouse design, rather one of mismatched tools. It was not initially well-understood that data warehouses were fundamentally different from other kinds of databases. Designers felt that if the data in the warehouse were properly organized, then it would perform properly – but less attention was paid to the database the warehouse was built on, which was often whatever OLTP database was handy – Oracle, DB2, etc.

It turns out that databases can be tuned in two different ways: They can be tuned to handle *many small transactions* – where "many" means perhaps millions per day, and "small" means affecting a handful of rows per transaction. This is what the databases sitting behind typical retail operations do. On the other hand, they can handle a *few large transactions* – where "a few" means perhaps a hundred per day, and "large" means affecting millions or even billions of rows per transaction. This first type of databases is called an *OLTP* (for *online transactional processing*) database. We'll call the second type a *data warehouse* (or *DW*) *platform*.

Databases are either good at OLTP or DW, but not both. Place sufficiently heavy data-warehouse-type demands on an OLTP database, and it will founder. If the database on the right in Figure 2 is an OLTP database, then there are going to be cases when large queries on its contents will fail.

This realization led to the introduction, beginning in the early 2000s, of *data warehouse appliances*, notably Netezza.

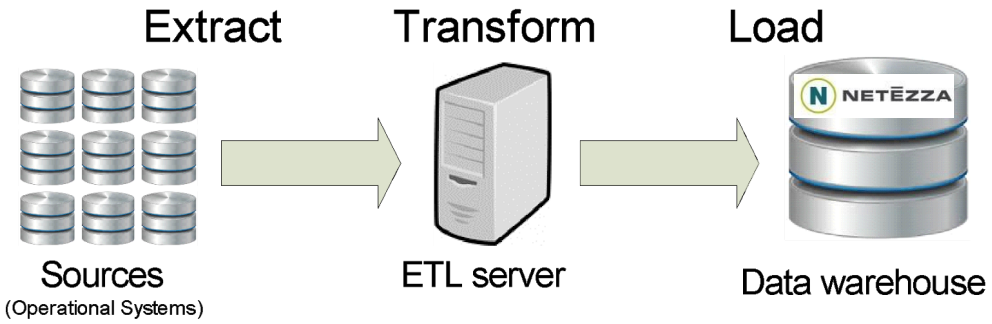

Figure 3

The Netezza platform was a parallel computer that could perform very fast SQL queries on large datasets and had other features that made it suitable for use as a data warehousing platform. Beginning in 2003, Netezza started being sold as a platform for data warehouses, often replacing Oracle as the data warehouse platform in Figure 2. In time, other solutions suitable for the right side of Figure 3 became available, such as Amazon Redshift. Apache Spark, a cluster platform, is one of the latest of these.

Some common advantages of these solutions over the OLTP databases they replaced are:

- They are much better at supporting "data-warehouse-type" queries – that is, joins or other queries involving millions or billions of rows.
- They are orders of magnitude less expensive.
- They are easier to maintain.
- They have a smaller footprint.
- They are not necessarily smaller in terms of actual storage.

The effect of these new technologies on the world of data warehousing was rapid and conspicuous; one would no longer find attempts to build serious DWs on OLTP databases.

But what about the rest of the picture? Do the changes on the right-hand of Figure 2 side have consequences for the ETL server as well? Indeed, they do since the ETL servers now represent an expensive *duplication of resources*. The technologies now used on the right-hand side of the diagram duplicate the capacities of the ETL server, which, in large part, consist of efficiently carrying out join- and sort-like operations on large datasets. The new DW platforms are, in effect, excellent transformation engines themselves. Why not use them to replace the ETL servers altogether? This suggests a simpler architecture, shown in Figure 4:

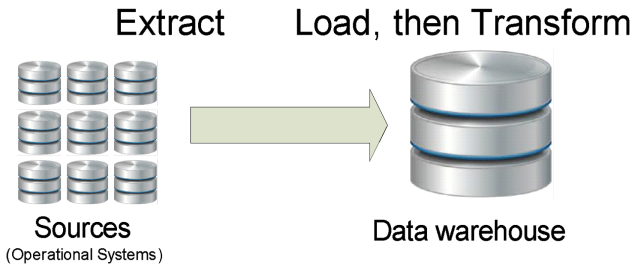

Figure 4

ELT: Nothing in-between

The advantages of the ELT architecture over ETL are difficult to overstate. They are best understood by simply comparing Figure 3 and Figure 4. When a Figure 4 architecture is possible, removing the ETL server confers the obvious benefits of reducing cost, reducing latency, and removing a data volume bottleneck. Obtaining these benefits would not only improve existing data warehouses; the reduced up-front cost, footprint, and maintenance overhead could lead to data warehousing resources being available to organizations for which they were previously out of reach.

There are, however, benefits of the ETL server that are difficult to do without. The ETL server is the platform for the ETL tools with which developers are familiar – tools like DataStage, Informatica, and Ab Initio. These tools relieve the developers of the necessity of writing code in SQL, Scala, C++ or whatever language of the platform might be, instead letting them program by constructing dataflows in a visual language. The tools also provide important abstractions such as “jobs”. “job sequences,” “job parameters,” and so on. For developers to home-grow their own tools for these tasks is laborious and error-prone. But if a data warehouse is to operate smoothly, reliably ingesting and processing data day after day, these tools are critical.

ELTMaestro for Spark

How, then, can we have our cake and eat it too? How can we get the architectural advantages of Figure 4 – dispensing with the ETL server, saving that money, and realizing those performance gains – without giving up the conveniences that tools like DataStage and Informatica provide? That is precisely the purpose of ELTMaestro.

ELTMaestro’s reason for existence is to *enable* the architecture shown in Figure 4. An edition of ELTMaestro exists for each supported platform that can be used on the right-hand side. We are proud, in this paper, to announce that these platforms now include Apache Spark.

Agents for secure extraction and communication

ETL servers have another role which we cannot ignore; they also extract data from sources. In cloud-based environments, extracted data may need to move across the open internet; this adds a dimension of security concerns that need to be addressed.

ELTMaestro extracts data from remote sources by using a scheme of *agents.* Agents are processes installed on data sources that know how to communicate with the target. Any number of agents may be installed on a source, and any number of agents can be installed overall. When a connection to a source is requested, the first responding agent that can support that connection is used. Multiplexing may occur when another connection to the same source is sought; in this case another, unused agent may reply and support the second connection. By supporting multiplexing, agents help the enterprise maintain a degree of control over bandwidth in a Cloud implementation.

Agents take care of encrypting and compressing data communication between source and target (using AES 256-bit encryption) alleviating the concern of eavesdropping.

Agents also help to maintain data security because the data sources where they reside tend to be inside internal networks where they are invisible to the outside world. The agent can see the target machine,

and can initiate communication with the target machine, but the target machine can't see the agent, other than to respond to its queries. Agents maintain contact with the target machine by pinging the target every few seconds, to see if there are requests for their data. If there are, a connection is established and data is sent to the target. The target never needs to know the IP address of the source, which helps to prevent illegitimate connections.

Conclusion

Apache Spark represents a potential revolution in data warehousing.

The cost and performance profile of Apache Spark means that data warehousing resources that were once only available to Fortune 500 companies with vast IT budgets are now available to very moderate-size firms with limited means.

Traditional ETL architecture is simply a stumbling block preventing this revolution. By enabling developers with the tools they need to build and maintain data warehouses with a platform architecture that makes sense, we are simply removing this stumbling block, and letting the revolution move forward.