PENTAHO DATA INTEGRATION

SCALING OUT
LARGE DATA VOLUME
PROCESSING IN THE CLOUD
OR ON PREMISE

A White Paper Presented by:

http://www.bayontechnologies.com
Open Source Business Intelligence Experts

Author:
Nicholas Goodman
Principal Consultant and Founder
ngoodman@bayontechnologies

Please post any questions, comments on this paper
to the Pentaho Forum

Introduction

Pentaho Data Integration (PDI) 3.2 provides advanced clustering and partitioning capabilities that allow organizations to “scale out” their Extract, Transform and Load (ETL) deployments instead of “scaling up” on expensive single node machines. Data volumes grow consistently but batch loading windows have remained the same creating challenges for any data warehouse. Scale out capabilities in PDI can reduce this risk in addition to being exceptionally cost effective compared to proprietary solutions. In this paper we'll review the performance of PDI scale out capabilities and it's ability to process billions of rows across as many as 40 nodes. This paper explores using Amazon Elastic Compute Cloud (EC2) as an infrastructure for building PDI clusters “on demand” demonstrating some compelling price / performance metrics for doing data processing in the cloud.

This paper answers the following questions:

Question 1 : Does PDI scale linearly when adding additional nodes to a cluster?

If it takes 4 hours to do our processing on 1 node will it take 1 hour to do it on 4 nodes? We want to see if we can simply add additional compute nodes to our cluster to decrease our batch processing time. It is desirable to have this linear scalability so that we can deploy the correct number of nodes to meet our batch time requirements.

Question 2 : Does PDI scale linearly when processing more data?

If we double the size of the dataset we're processing, does PDI take twice as long to process that data? We want to see if a PDI cluster can handle increased amounts of data gracefully.

Question 3 : What are the key price and performance metrics of ETL in the cloud?

We want to see some overall performance and cost numbers from our cloud experiments. We will eventually want to compare this to a physical in house deployment scenario but for now we just want to get some high level metrics such as “$ per billion rows sorted and aggregated.”

Question 4: What does someone deploying PDI on Amazon EC2 need to know?

We want to develop a set of well known issues and any best practices that can be used to deploy PDI clusters on the cloud. What information is helpful for these deployments?

In addition to providing summarized results and analysis of these four questions, there are several sections that explain in greater detail the specific test cases, datasets, and results from experiments performed on EC2. The PDI transformations and test dataset generators are available for download to explore the test cases and results presented here (see Appendix B – References for the download link).

Pentaho Data Integration (PDI) Clusters

PDI clusters are built for increasing performance and throughput of data transformations; in particular they are built to perform classic “divide and conquer” processing of datasets in parallel. Clustering capabilities have been in PDI since version 2.4, with new features being added with every release. The new cluster capabilities in PDI 3.2 improve the ability to dynamically create clusters of available servers, greatly reducing the amount of work necessary to manage a PDI cluster at runtime. These "dynamic cluster" capabilities make the cloud deployments we discuss in this paper far more simple and straightforward to manage.

PDI clusters have a strong master/slave topology. There is ONE master in cluster but there can be many slaves. Transformations are "broken" into master/slaves at run time and deployed to all servers in a cluster. Each server in the cluster is a running application called "Carte." Carte is a small PDI server running a very lightweight web service interface to receive, execute, and monitor transformations.

PDI clusters are not, however, intended as a "high availability" or "job management" system to handle fault tolerance and load balancing. The PDI clustering and partitioning mechanisms are relatively naïve at runtime; PDI clusters do not do any optimizations based on CPU/load or observed performance of nodes in the cluster.

PDI clustering is a capability in the software; it is not dependent on any particular cloud vendors API/implementation. It can just as easily, and regularly is, deployed on internal dedicated hardware.

Amazon EC2

EC2 allows for dynamic provisioning of compute resources such as servers, block devices, IP addresses, etc. This infrastructure allows for companies to provision machines when they need the compute capabilities and then release those resources when they are no longer needed. The rate for services is tied only to use and you only pay for what you use. The price of using 100 machines for 1 hour is the same as using 1 machine for 100 hours.

In addition to providing raw compute resources (servers), EC2 also provides a block device called Elastic Block Storage (EBS). This service acts like a Storage Area Network (SAN) for EC2 that allows block devices to be mounted to EC2 servers and filesystems to be created and used.

Since you pay for only what you use with no long term hardware commitments this can be a very attractive model for those who need a lot of compute resources for short periods of time. Consider some of these approximate costs:

- 40 small servers (1.7 GB of memory with 1 virtual core) for 1 hour = $4.00
- 20 TB of block storage used for 1 hour = $2.80

ETL and Cloud : A Superb Match

This profile of compute resource needs is an ideal match for nightly, weekly and monthly ETL batch processing which typically needs a surge of compute resources during the batch window. Daily batch ETL processing usually kicks off in the middle of the night, and processes the settled data from the previous day. The batch window is the amount of time from the beginning of the process (data available to process) to the time when the data processing is completed (reports are ready for users). The batch window remains relatively fixed (you can start processing at 12:01 am and people look at reports at 8:00am) but the volume of data for processing is growing at a significant rates in many organizations.

The attractive cloud computing costs coupled with the periodic surge of ETL compute needs makes the cloud an ideal location for deployment.

Experiments

Test Data : TPC-H

We chose to use the TPC-H data set from the Transaction Processing Council. We chose this dataset for the following reasons:

- TPC provides a data generator that allows generating different scales of data. In our experiments, we generated THREE datasets. Scales **50, 100, 300** (in GB).
- TPC-H is a relatively well known DSS dataset. Rather than invest time and effort in building our test dataset we could use something that is a relatively known dataset for data warehouse database benchmarks.
- It's a test dataset that is independent of these tests; readers can be guaranteed this dataset wasn't intentionally built to perform well with these tests.

Please Appendix B – References for links to TPC-H documents that contain specifics of the datafiles we used in the experiments. These documents also include a diagram of the relationships between the files (similar to an ER diagram).

The "lineitem" file is the main set of transaction records and is the largest in the TPC-H dataset. It contains information on items purchased in a retail store. There are some relatinships that will be used in our experiments:

- Lineitems have a "party/supplier" dual value foreign key to the Part / Supplier data files.
- Suppliers have a foreign key to the country of origin.

The files are "flat" files with varied data types (characters, integers, and numbers) that are delimited by a pipe (|). The data files used in the experiments were not modified in any way after generation using the TPC provided "dbgen" program.

File	# of Fields	Scale	Size in GB	# of Lines
lineitem.tbl	16	50	36.82	300,005,811
lineitem.tbl	16	100	74.12	600,037,902
lineitem.tbl	16	300	225.14	1,799,989,091
supplier.tbl	7	50	0.07	500,000
supplier.tbl	7	100	0.13	1,000,000
supplier.tbl	7	300	0.40	3,000,000
nation.tbl	4	ALL	0.00	24

NOTE: This experiment is in NO WAY to be an official TPC-H benchmark. TPC-H Benchmarks have very specific requirements for databases performing them. This experiment does none of these things.

ETL Scenarios

In order to answer our experiment questions, we needed ETL that:

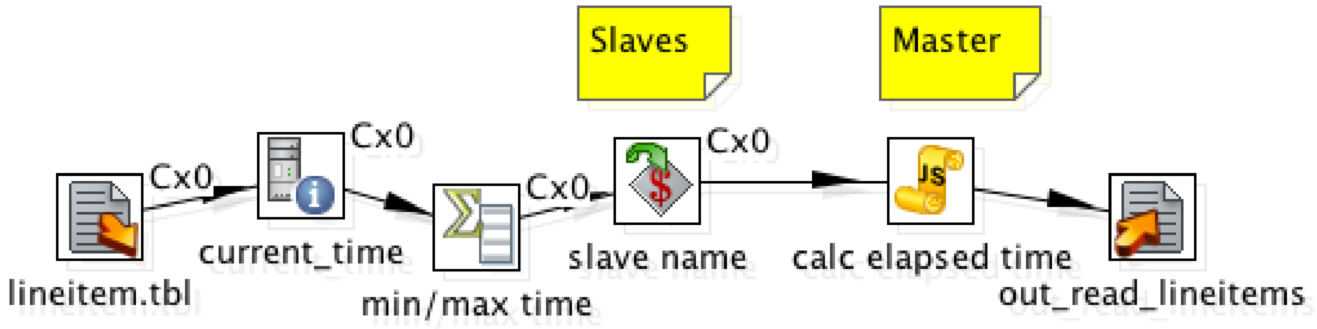
- Was relatively simple so readers not familiar with PDI would still be able to understand the overall ETL processing.

- Did some non trivial processing.
- Matched to a relatively common use case.

Several different ETL scenarios were evaluated, and eventually two ETL transformations were tested. Some high level explanations of the transformations are presented here, along with full screenshots in Appendix A.

READ (read_lineitems_lazy.ktr)

READ is a baseline to see how fast the cluster can simply read the entire file. It reads records and does no additional processing on those records. READ ends the slave portion of the transform and then returns some information on how this particular slave server performed.



READ won't map to a real world use case; it's simply used for us to evaluate the infrastructure and get a baseline for reading large data files.

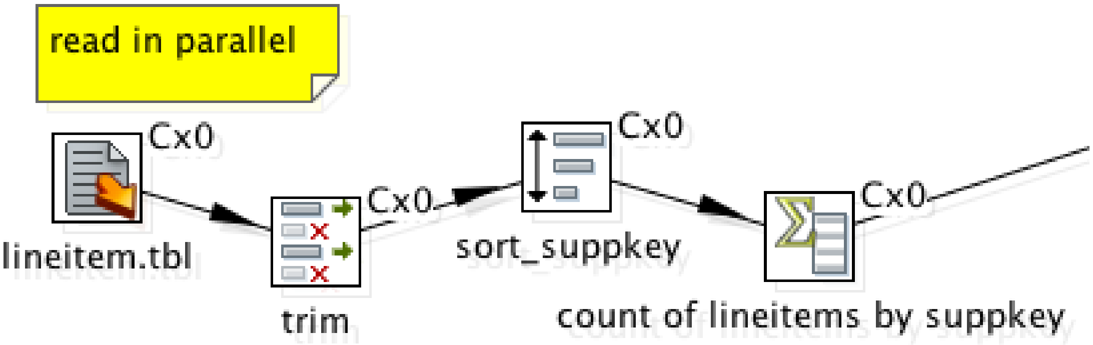
READ OUTPUT FILE (out_read_lineitems)

READ outputs some useful information about individual node performance on the server (out_read_lineitems). It includes the server name, the # of records processed, and the per server throughput.

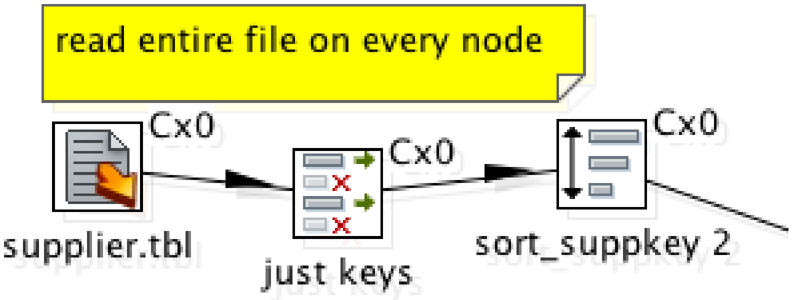
Slave	# of Records	Elapsed Time (s)	Throughput/node
Dynamic slave [10.250.15.69:8080]	30144428	894	33718.60
Dynamic slave [10.251.199.67:8080]	29916897	901	33204.10
Dynamic slave [10.250.19.102:8080]	29917123	903	33130.81
Dynamic slave [10.250.47.5:8080]	30144739	908	33199.05
Dynamic slave [10.251.42.162:8080]	29988406	920	32596.09
Dynamic slave [10.251.71.159:8080]	29916947	942	31758.97
Dynamic slave [10.251.122.49:8080]	29915685	949	31523.38
Dynamic slave [10.250.11.95:8080]	29916158	950	31490.69
Dynamic slave [10.251.203.114:8080]	29916164	1044	28655.33
Dynamic slave [10.250.137.204:8080]	30229264	1095	27606.63

SORT (sort_lineitems_group_by_nationkey.ktr)

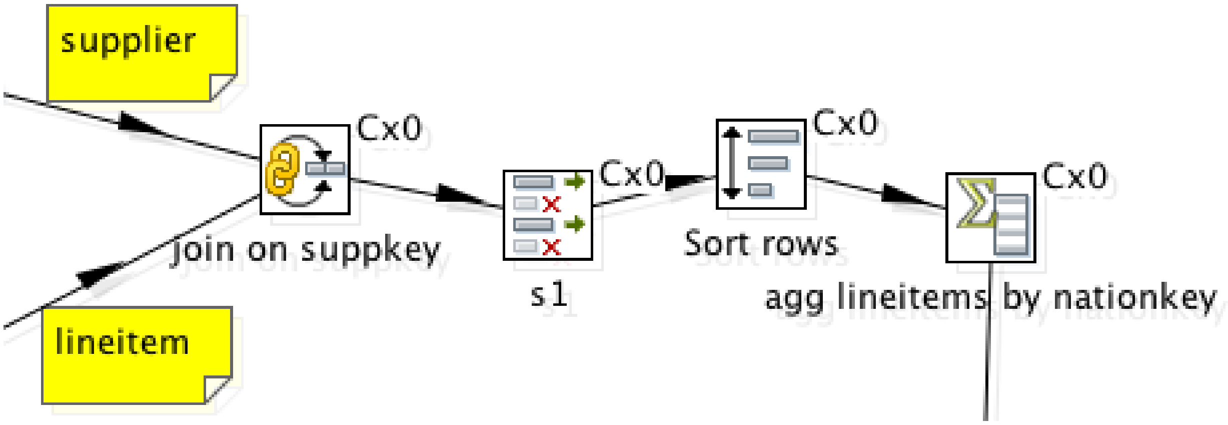
SORT represents a real world need for sorting, joining, and aggregating large transaction files. SORT is similar to READ in that it reads the datafile but then it SORTs and aggregates the file to build a "COUNT OF LINEITEMS by SUPPLIER" total.



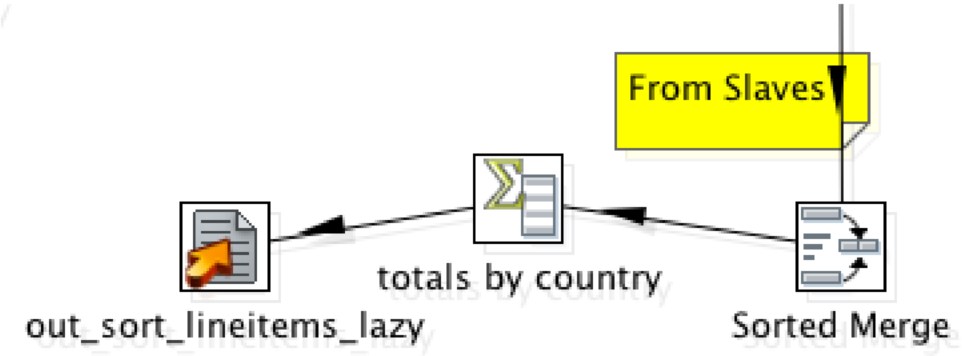
SORT is also doing another read and sort of another large file (supplier). In this case, there is no aggregation but the data is sorted to join it with the "COUNT OF LINEITEMS by SUPPLIER" branch.



SORT then does a join with another large file (supplier) using the shared "suppkey" value from both files. After matching the "COUNT OF LINEITEMS by SUPPLIER" to the full supplier record we sort/aggregate by nationkey.



We aggregate the results (to build combined totals) on the master and output the result to a datafile on the master server.



SORT OUTPUT FILE (out_sort_lineitems_group_by_nationkey)

The output is a summary of the number lineitems by supplier country, a simple 25 line file with one record per nation and the total # of records in the lineitem data file. For instance, for the TPC-H scale 300 file the output file looks like this:

Nation Key	Count of Lineitems
0	71886458
1	71969681
2	72038468
3	72179079
4	72209741
5	72146529
6	72178370
7	72055521
8	72184094
9	71870036
10	71974851
11	72051132
12	71627623
13	71529955
14	71795817
15	72079318
16	71897393
17	71862655
18	71974391
19	72037468
20	72201477
21	72056815

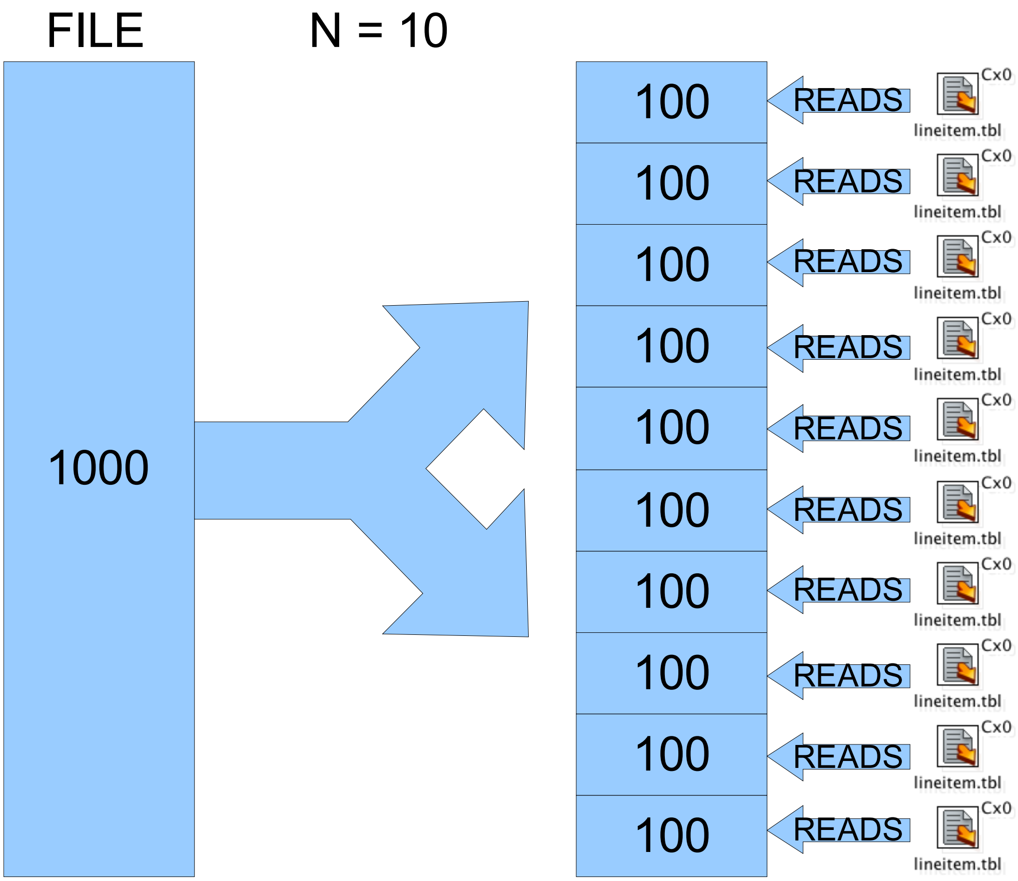
22	71873005
23	72432871
24	71876343

With each test, we verified we had the correct results in the files. This included checking row counts (both READ and SORT) and results (SORT).

Parallel File Processing

The designation of Cx0 indicates that this step will be run on the cluster. By default, most steps don't need any additional cluster configuration. Most discreet units of work (Calculations, decodes, filtering, etc) work independently and need no special cluster configuration but inputs and outputs, if run on the cluster, need to be configured to operate on a portion of the data.

The CSV Input Step has a configuration parameter "Running in Parallel?" which will instruct the step to read only it's "segment" of the file if deployed on a cluster. This enables each node to read approximately 1 / N part of a file.



This also means that for "Running in Parallel?" all nodes need read access to the same file (lineitem.tbl) at the same location (/data) to each have the chance to process a different section of the file.

EC2 Cluster

Instance Image (AMI)

EC2 provides three instance sizes: small, large, xlarge. Small is a 32 bit image, Large and X-Large are 64 bit images. In this experiment we used small instances, with the following hardware specifications:

- 1.7 GB of memory
- 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)
- 160 GB of instance storage, 32-bit platform

We built our own machine server image (AMI), based on Fedora 8. This machine image was built using the standard Amazon EC2 build tools. The machine was built to accept "user data" at startup time that serves to "configure" the machine at startup. The AMI image remained static (untouched) throughout this experiment but we delivered varied slave server configuration values when we started sets of servers.

We passed in the following configurable parameters when we started our servers:

- pdi-distribution-url
  The URL of where to get PDI. The AMI downloads and dynamically installs PDI so that upgrading versions of PDI does not require changing the AMI.
  Example: http://www.bayontechnologies.com/pdi-ce-3.2.0-RC1-r10482.zip
- carte-slave-server-xml
  This is the XML that the AMI will start "Carte" with. For the MASTER this contains no specializations. For the SLAVE servers, this needs to contain the hostname of the MASTER so that the SLAVE servers can register themselves with the MASTER.
  Example:
  ```
  <slave_config> <slaveserver> <name>carte-slave</name>
  <hostname>localhost</hostname> <network_interface>eth0</network_interface>  <!--
  OPTIONAL --> <port>8080</port> <username>cluster</username>
  <password>cluster</password> <master>N</master> </slaveserver> </slave_config>
  ```
- data-volume-device
  The device location of the EBS block device that has been attached to this instance.
  Example:
  /dev/sdl
- data-volume-mount-point
  The location to mount the EBS block device.
  Example:
  /data
- java-memory-options
  Memory options for starting Carte
  -Xmx1024m

This "user data" is passed to EC2 as part of the request to start servers.

EBS Volumes

EBS was the mechanism which provided all nodes in the cluster the ability to read parts of same file. EBS volumes can be attached to only one running server at a time and can range in size from 1 to 1000 GB.

We created a 1000 GB EBS volume and installed an XFS filesystem on it. XFS was used instead of ext3 to enable large file (> 2.4GB) support. We generated three different scales of TPC-H data onto this same volume so any scale could be run from mounting a single volume.

Since an EBS volume can only be attached to one server at a time, we needed a way to quickly clone and mount individual volumes to running instances. We took a snapshot of our volume using EBS snapshot capabilities. EBS also allows for quick creation of new volumes from snapshots so we can quickly create N volumes, one for each of our N nodes that has the dataset on it.

Scripts

We built scripts to help manage the life cycle of a PDI EC2 cluster. This included major functions such as:

- Creating Master and Slave Instances
- Creating Master and Slave Volumes from the TPC-H dataset snapshot
- Attaching the Volumes to Instances
- Status Reporting on all Instances
- Executing of clustered transformations

A PDI cluster of N machines was CREATED using the following process:

- Master Volume was created from snapshot (ec2-create-volume)
- N Volumes were created from snapshot (ec2-create-volume)
- Master Slave was started (ec2-run-instance)
- Once Master was running, attach volume (ec2-attach-volume)
- N Slaves were started (ec2-run-instance)
- N Volumes were attached to N slaves (ec2-attach-volume)
- Slaves automatically registered themselves with Master

A PDI cluster was DESTROYED using the following process:

- Slave Instances were terminated (ec2-terminate-instances)
- Slave Volumes were detached (ec2-detach-volume)
- Master Instance was terminate (ec2-terminate-instances)
- Master Volume was detached (ec2-detach-volume)
- Slave Volumes were deleted (ec2-delete-volume)
- Master Volume was deleted (ec2-delete-volume)

Using the lifecycle scripts, clusters of 10, 20, and 40 machines were CREATED and DESTROYED in minutes.

NOTE: The scripts include private security credentials; they are not included in the pdi_scale_out_benchmark_kit.zip.

Results

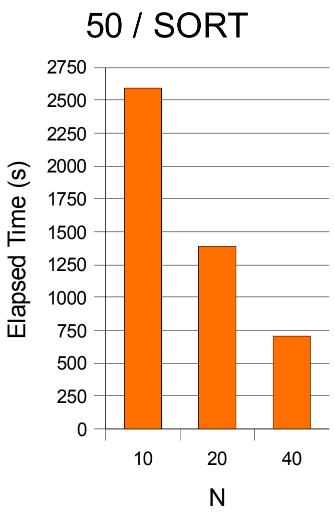
We ran the experiments for all three scales (50, 100, 300) using three different cluster sizes (10, 20, 40) for both SORT and READ. The results are summarized in the following table.

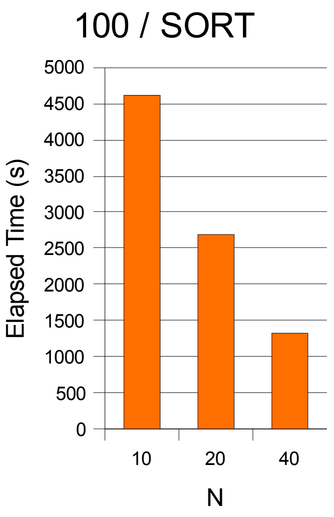
Scale	# of Lines	Nodes (N)	Elapsed Time	Rows / sec	Transform
50	300005811	10	1095	273978	READ
50	300005811	20	836	358859	READ
50	300005811	40	640	468759	READ
100	600037902	10	1787	335779	READ
100	600037902	20	1736	345644	READ
100	600037902	40	1158	518167	READ
300	1799989091	10	6150	292681	READ
300	1799989091	20	5460	329668	READ
300	1799989091	40	3835	469358	READ
50	300005811	10	2588	115922	SORT
50	300005811	20	1389	215987	SORT
50	300005811	40	704	426145	SORT
100	600037902	10	4615	130019	SORT
100	600037902	20	2680	223895	SORT
100	600037902	40	1318	455264	SORT
300	1799989091	10	15252	118017	SORT
300	1799989091	20	8390	214540	SORT
300	1799989091	40	4014	448428	SORT

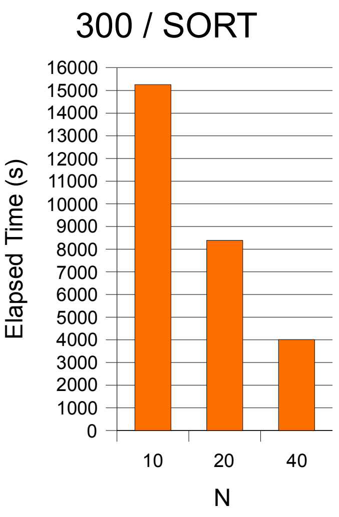
The full results of the experiments are included in an Excel file in the Benchmark Kit and are not presented in their entirety here. We will examine the test results in the context of our experiment questions.

Question 1 : Does PDI scale linearly when adding additional nodes to a cluster?

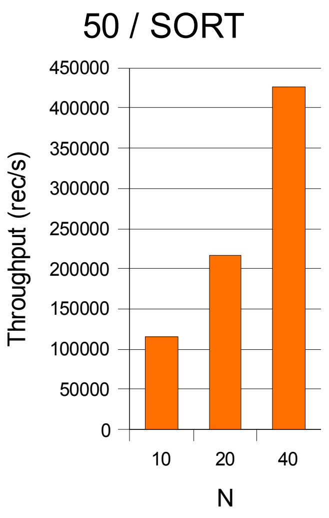
SORT: Elapsed Time

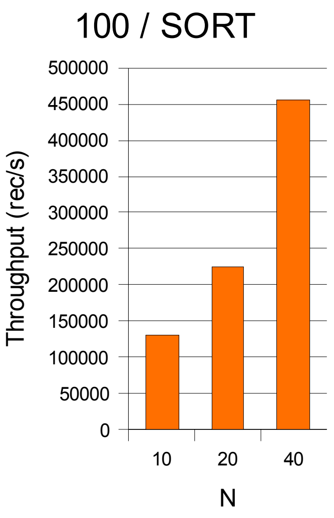


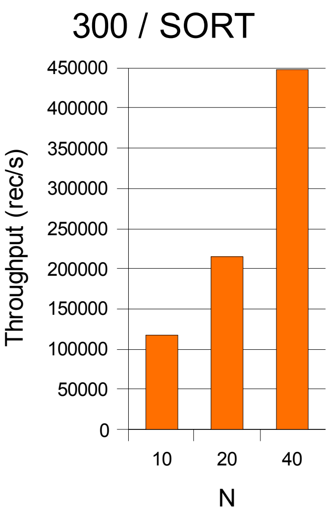




SORT: Throughput (rec/s)

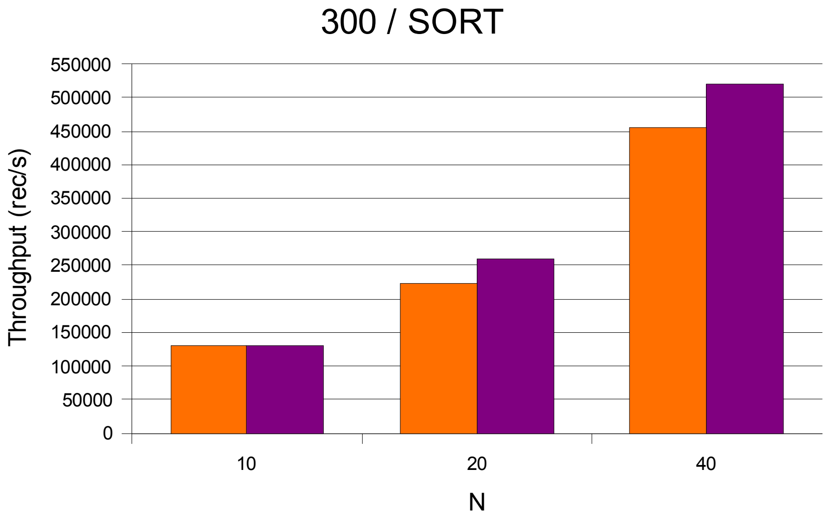






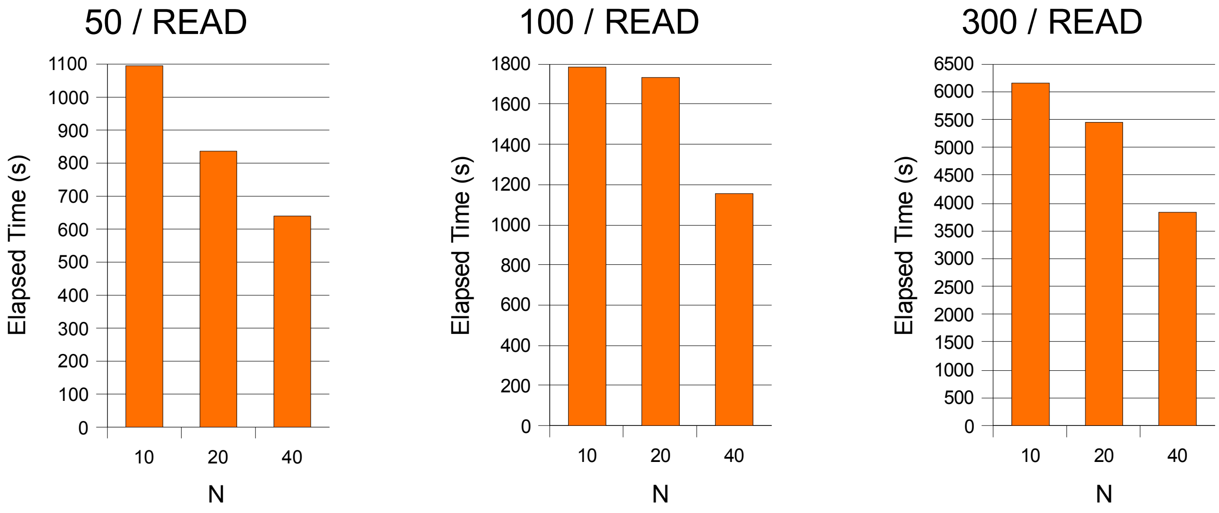
The results from SORT fit the assumption that adding more nodes to a cluster improves throughput and overall processing close to linear but not exactly.

Consider the 100 / SORT across the three cluster sizes. If we start with the baseline of N = 10 then we can project out how long 20, and 40 should take (N 10 * 2, and N 10 * 4) to come up with the projected pure linear scaling numbers.

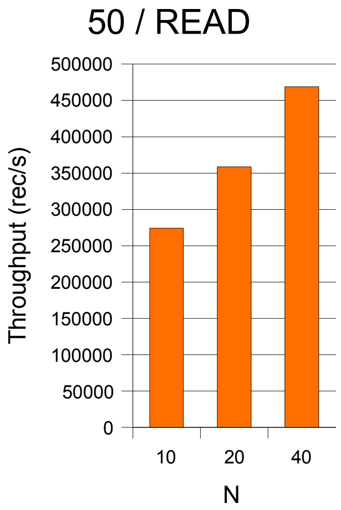


Our performance doesn't match our projections exactly, and does start to show some diminishing returns on adding more nodes. Some of this is likely to be variations in cloud compute resources (discussed in Question 4 results) since we had some samples that actually BEAT their projected performance. For example, 300 SORT / N40 was faster than the projected N10 * 4 number indicating that there were variations on underlying hardware conditions during the experiments. However, the data indicates there is not a direct linear relationship on adding more nodes.

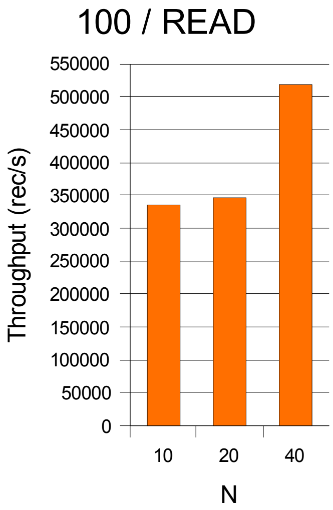
READ: Elapsed Time



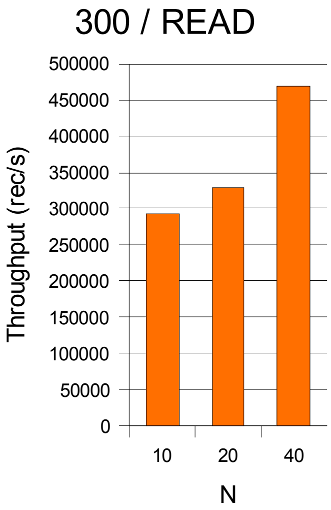
READ: Throughput (rec/s)



50 / READ



100 / READ



300 / READ

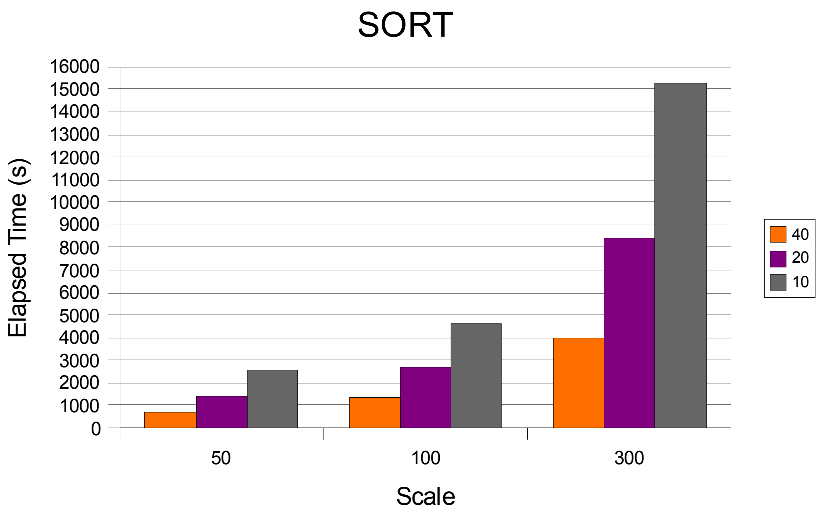
The performance of N40 isn't even close to the projected N10 * 4. In fact, N40 never even shows more than twice the performance of N10.

On the READ transformation, there is no inter-node communication and the nodes all operate independently on a different section of the file. There is no real processing and only a small amount of data (a few Kb) is passed from the slaves to the master. This seems to indicate that our underlying storage system (EBS) wasn't scaling linearly on read performance to volumes based on the same snapshot. We did not, unfortunately, do any sort of OS level benchmarks on the mounted volumes for raw performance on EBS outside of PDI which would have been able to determine if this was indeed accurate.

ANSWER: Our experiment was not a 100% controlled environment and we think our underlying cloud infrastructure skewed the results. Even so, our SORT (which will throttle I/O a bit) showed close to linear scaling and significantly improved performance when adding new nodes. **PDI on EC2 scales nearly linearly, but not exactly likely due to EBS variations.**

Question 2 : Does PDI scale linearly when processing more data?

The results for different scales and nodes have already been presented. Let's look at the performance of SORT to see how well it scaled with the increasing scales (50, 100, 300).



The values for all node sizes at 100 are approximately 50 * 2. The values for all node sizes at 300 are approximately 50 * 6. We observed better than linear results on some values which also indicate a variation in underlying compute resources/EBS performance.

ANSWER: PDI scales linearly with increasing data sizes in the cluster sizes and data scales we tested.

Question 3 : What are the key price and performance metrics of ETL in the cloud?

Let's first arrive at some overall performance metrics for our cluster. We looked at the overall SORT performance for each node for all three scales and arrived at an average, per node SORT performance of 11,373 records per second. This is the average sort performance on our relatively linear scaling transformation which means we can use this figure with relative confidence at different scales (data sizes) and cluster sizes (instances in cluster).

Nodes	Throughput	Throughput / Node
40	443278.79	11081.97
20	218140.49	10907.02
10	121319.17	12131.92
Average SORT throughput / node		11373.64

Here is the loaded cost of our “small” EC2 instance:

EC2 Cost	Cost / Hour
Server Time	$0.10
EBS Volume (1000 GB)	$0.14
EBS Load from S3	$0.01
Bandwidth	$0.00
TOTAL	$0.25

How many rows can we SORT in 1 hour on one node?

11373 X 60 (seconds) X 60 (minutes) = **40,942,800 Sorted Rows / Hour / Small Node**

What is the price per billion rows processed on the cloud?

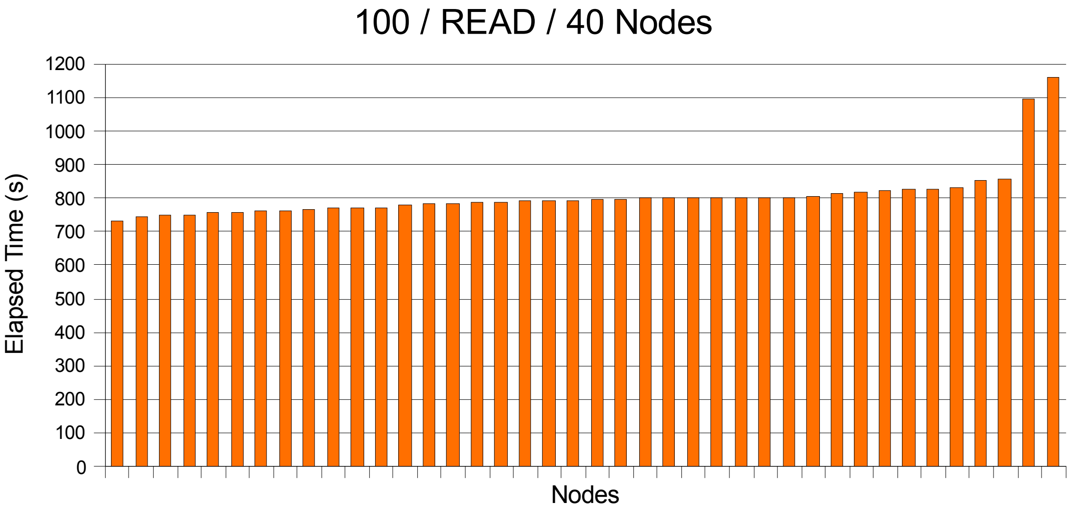
($0.24 * 1000000000) / 40,942,800 = **$5.86 compute cost to SORT a billion rows on EC2**

NOTE: These calculations do not take into account cluster startup time (approximately 20 minutes for 40 node cluster) and smaller negligible bandwidth costs.

Question 4: What does someone deploying PDI on Amazon EC2 need to know?

All Instances are Not Created Equal

Our READ transformation produced individual node performance metrics. We observed some significant variations on individual node performance. It is unclear if the difference in individual performance was the instance (cause by other virtual machines on the same physical server) or it's connection to EBS (reading the file) but we saw some nodes taking up to 58% longer than others even though they were processing a similar size/number of rows.



Most are relatively smooth, with nodes 1-38 completing within 100 seconds of each other. Nodes 39 and 40 toke significantly longer causing the overall performance of the transformation to be equal to the "slowest" node. The full test results show the averages on a per node basis of throughput which gives a more accurate per node performance number but is irrelevant for practical use. **PDI clusters are only as fast as their slowest node.**

PDI assumes a homogenous set of compute resources. It's division of file segments and allocation to nodes does not take into account any variations in performance or sizing of the machines. Systems like MapReduce / Hadoop have built in an optimizations to accommodate this and reallocates chunks of works to different servers based on load and failures. PDI has no such capabilities.

EBS Volume Performance

Based on our READ performance we observed EBS performance degradation with more nodes reading from a volume created from a snapshot. This is likely due to the fact that EBS behind the scenes loads the snapshot from S3 asynchronously and we were hitting blocks ahead of them being loaded. Each EBS volume is hitting different parts of the files and thus would be hitting different EBS/S3 blocks.

We should do a follow on experiment of parallel read performance from S3 directly, bypassing EBS.

Private IPs vs Public IPs

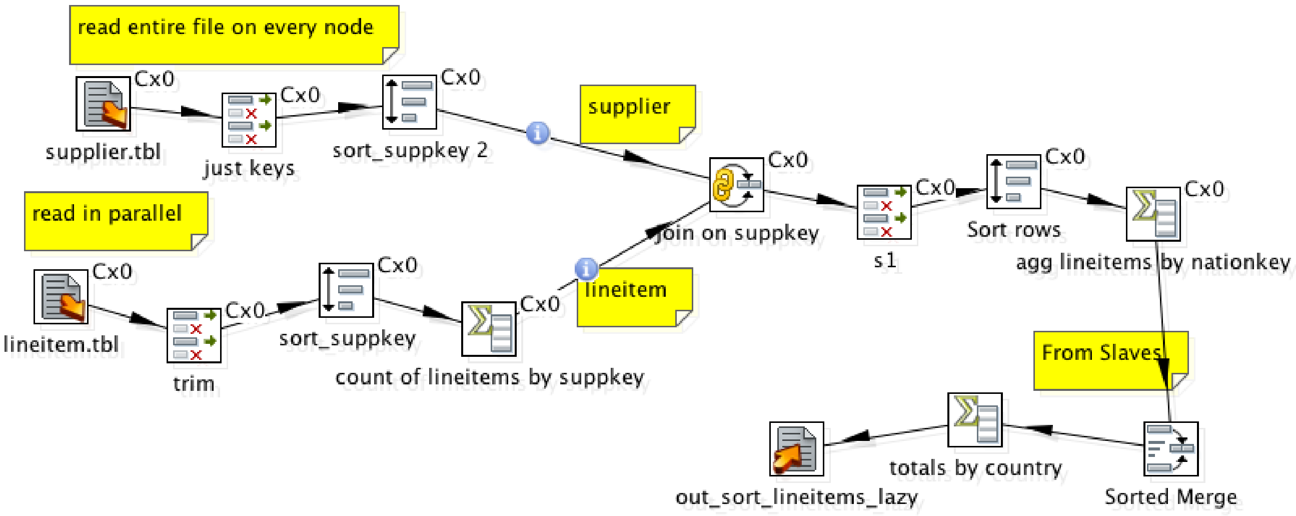
Amazon EC2 has two IP addresses for every machine; a private and a public. Private IP is used to communicate internally within Amazon EC2 networks and is believed to be the faster of the two networks. Public IP is used to communicate with the outside world.

In our cluster startup/monitoring scripts we needed to use the Public IP to communicate with the server but we wanted the servers to communicate with each other using their Private IPs. In addition to being faster, there are no bandwidth charges for using the Private IP while there is an Amazon EC2 charge for using the Public IP – even if it's within EC2 networks. This prevents us from running our clustered transformation directly via the PDI visual designer to the remote servers; we created a wrapper Job and shipped it to the master and ran it from the command line there. PDI 3.2 has the ability to export and ship the job, but it wasn't available on the command line "kitchen.sh" when we ran these experiments.

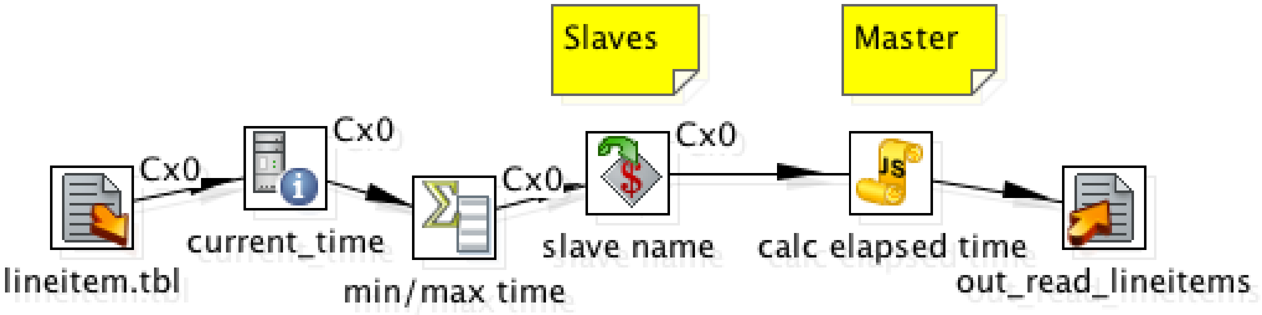
Readers should take care to setup their PDI cluster to use private IP addresses for inter node communication to save money and maximize network performance.

Appendix A – Test Transformations

sort_lineitems_group_by_nationkey.ktr is available in pdi_scale_out_benchmark_kit.zip



read_lineitems_lazy.ktr is available in pdi_scale_out_benchmark_kit.zip



Appendix B – References

- Benchmark Kit
  http://www.bayontechnologies.com/bt/ourwork/pdi_scale_out_benchmark_kit.zip
- TPC-H
  http://www.tpc.org/tpch/default.asp
  TPC-H Data generator
  http://www.tpc.org/tpch/spec/tpch_2_8_0.tar.gz
  TPC-H Document
  http://www.tpc.org/tpch/spec/tpch2.8.0.pdf
- Amazon EC2
  http://aws.amazon.com/ec2/
  EBS
  http://aws.amazon.com/ebs/
- PDI
  http://kettle.pentaho.org/
  Dynamic Clusters
  http://wiki.pentaho.com/display/EAI/Dynamic+clusters