White paper
Intel® CoFluent™ Studio
Intel® CoFluent™ Technology for Big Data

Optimize configuration parameters faster and more accurately, and speed up analyses of scaling big-data clusters

Intel and Dell have collaborated on research to help developers better optimize big data clusters. Our research shows that the workload performance is CPU-sensitive and sensitive to scaling the number of nodes in a cluster. Accurate simulations of these workloads provide a development tool for choosing better values for configuration parameters. This can help both expert and less experienced developers save development time, improve capacity planning, and accurately tune big data clusters for business needs.

Authors

Kebing Wang
Intel Corporation
kebing.wang@intel.com

Bianny Bian
Intel Corporation
bianny.bian@intel.com

Antonio Cisternino
Dell Innovation Center
at the University of Pisa
cisterni@di.unipi.it

Mike Riess
Intel Corporation
mike.riess@intel.com

Introduction

The TPCx-BB Express Benchmark (TPCx-BB)*[1] is designed to measure the performance of big data analytics systems. The benchmark contains 30 use cases (queries) that simulate big data processing, big data storage, big data analytics, and reporting. The benchmark allows developers to explore efficient ways to optimize configuration parameters for those 30 representative workloads. Better optimizations could help developers speed up analyses of scaling clusters. Such optimizations could also help them select the best components earlier in the development cycle for business needs; and predict the best parameter optimizations for future generations of hardware and software.

The project team for our optimization research was a collaboration between Intel and Dell. For this optimization project, our team used the TPCx-BB benchmark to evaluate the performance of both software and hardware components for big data clusters. We used the TPCx-BB benchmark because we find that it has good coverage for different data types. The benchmark also provides enough scalability to address challenges of scaling data size and nodes. We have gained key insights into designing big data analytic systems by using TPCx-BB.

Table of Contents

We do need more than TPCx-BB to evaluate and design complete, end-to-end big data systems. This is because there is a difference between an analytics system and a real-world, end-to-end system. For example, the data flow of an end-to-end system should include data ingestion.

Data ingestion moves data from where it originates in a system (such as Apache Hadoop*) to where it can be stored and analyzed. Importing that data at a reasonable speed can be challenging for businesses that want to maintain a competitive advantage.

However, TPCx-BB was not designed to evaluate the performance of software and hardware for data ingestion. Unfortunately, with TPCx-BB, there is a strict limitation on bandwidth and latency for real-time processing.

This paper discusses our experiences and lessons learned using TPCx-BB to evaluate the performance of software and hardware for real-time processing. We then offer advice on how to extend TPCx-BB to evaluate data ingestion and real-time processing. Finally, we share some ideas on how to implement fuller TPCx-BB coverage for end-to-end big data clusters.

In this project, we compared the optimized parameter values suggested by Intel® CoFluent™ Technology for Big Data (Intel® CoFluent™), to the settings chosen by big-data experts. Results showed that Intel CoFluent delivered a 32% gain in the benchmark performance score over the parameter choices of expert human developers. This is an improvement equivalent to the performance gain typically seen from a new processor generation.

A scaled analysis of benchmark workloads shows that TPCx-BB queries are definitely sensitive to processor performance. For example, the scaled analysis showed that upgrading the processor resulted in an average decrease in execution time of 20%. The analysis also showed that queries with a large input size can benefit from having more nodes in the design. For example, scaling by a factor of 2 resulted in an average 33% decrease in execution time for those types of workloads.

These and other workload-specific results show that expert developers can use simulations to better optimize the values of configuration parameters for big data clusters. Accurate optimizations can also help less experienced developers understand design options better, and gain confidence in optimizing their clusters for specific workloads. With better optimizations, all developers can choose better values for configuration parameters, save development time, improve capacity planning, and optimize their big data clusters more accurately for specific workloads.

This paper describes our test environment, results, and insights for optimizing big data clusters. We also provide our data on the performance of specific workloads, based on upgrading the processor, upgrading the Ethernet, or scaling the number of nodes in the configuration.

Project scope and tools

Our project had three main elements: the physical big-data test setup, the industry-standard TPCx-BB benchmark with its 30 queries (workloads), and the Intel CoFluent modeling and simulation tool. With these project elements in place, we addressed the issue of helping developers optimize the hundreds or even thousands of configuration parameters they often face when configuring a big data cluster.

Big data clusters and Apache Hadoop*

Optimizing software parameters for big data clusters is a complex challenge. A major reason for this challenge is Apache Hadoop,* which is a complex open system used to drive many big data projects. Already, big data is becoming a synonym for Hadoop.

As a complex system, Hadoop has many software modules that process large quantities of data in a distributed way. A critical issue in using Hadoop is that each Hadoop software module has hundreds of configuration parameters. For example, Apache Hadoop YARN* has almost 400 parameters in its default configuration file.

The performance of a big data cluster can depend significantly on the values chosen for these thousands of parameters. However, for convenience, developers often simply use default parameter values. This is because it takes a high degree of experience and skill, as well as time and resources, to identify cluster-specific values that would provide better, optimized performance.

TPCx-BB Express Benchmark* as an industry standard

As big data technologies become widely adopted, there is a growing need for an industry-standard benchmark to evaluate and compare the performance of these systems. Such a benchmark should address the entire data flow. It should also cover as many big data use cases as possible. Based on our research into big data cluster deployment, optimization, and performance, we propose using TPCx-BB as a performance benchmark.

TPCx-BB is based on TPC BigBench*[2], which is a framework for end-to-end big data analytics. TPCx-BB is designed to measure the performance of big data systems. In particular, TPCx-BB addresses the variety, velocity, and volume aspects of big data systems that contain structured, semi-structured, and unstructured data.

One of the advantages of TPCx-BB is that it contains 30 queries (use cases, or workloads) that simulate big data processing, big data storage, big data analytics, and reporting. These use cases cover different but representative categories of big data analytics from a business perspective. Developers can use this benchmark to evaluate the performance of both software and hardware components for their big data systems.

Based on our research, we suggest TPCx-BB as an industry standard because TPCx-BB has good coverage of these different data types. In addition, TPCx-BB provides enough scalability to address big data issues typically seen with data size and node scaling. Also, the benchmark has enough workloads so that, in this project, we were able to identify the best optimizations for use cases based on workload category.

Intel® CoFluent™ Studio

Intel® CoFluent™ Studio is a powerful software package for modeling and simulation. Intel® CoFluent™ Technology for Big Data is a customized version of Intel CoFluent Studio.

For big data workloads, Intel CoFluent Technology for Big Data defines a big data workload and software stack, including tables, schema, indexing, distribution, operations, and other aspects of a big data model.

This customized version of Intel CoFluent Studio maps that big data software stack and workload to a hardware-based model. The customized tool then defines a big data cloud-based topology, including network, processors, memory, disk type, and so on.

The customizations of Intel CoFluent Technology for Big Data make it easier for developers to model and simulate their big data workloads, and identify optimal configurations for their business needs.

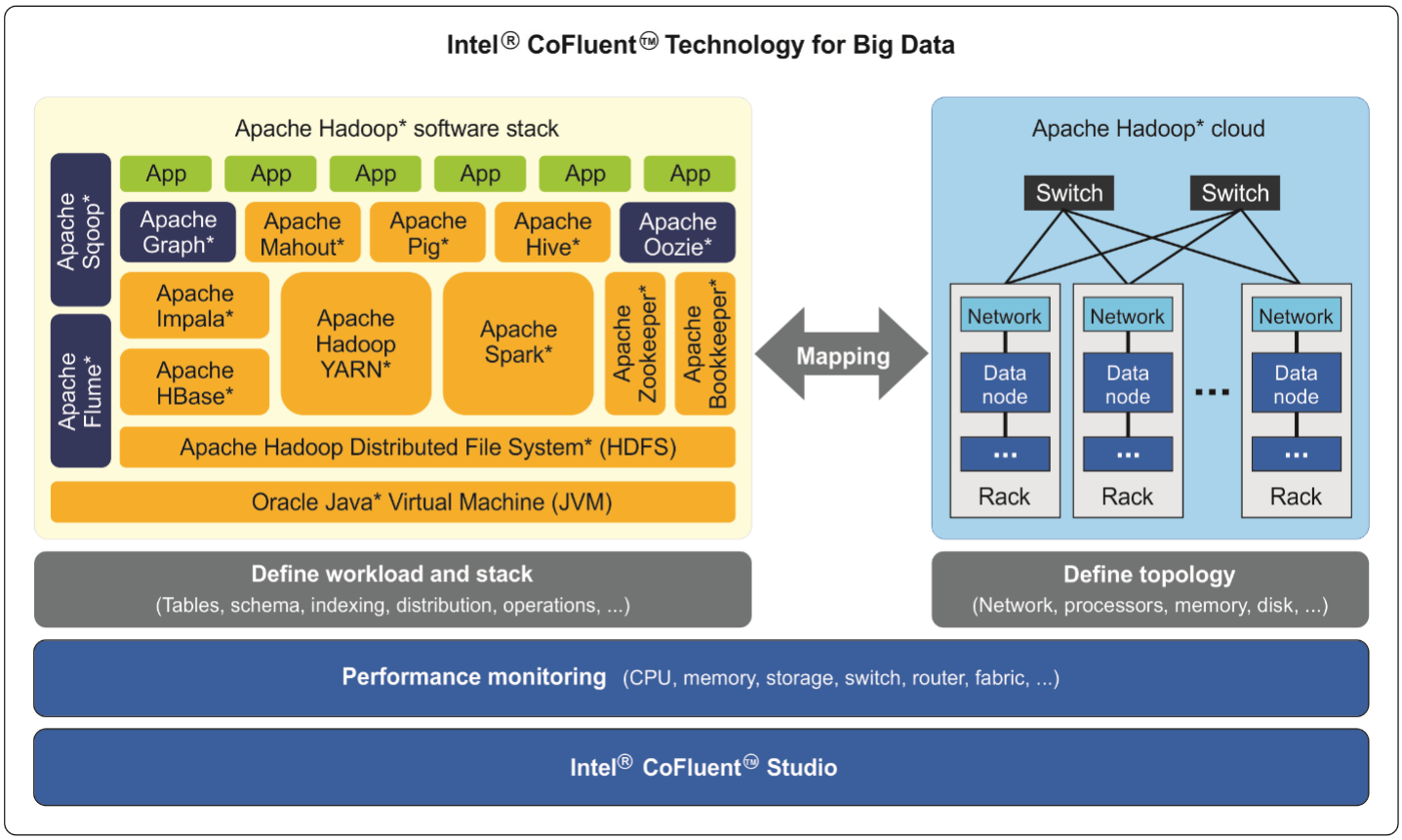

Figure 1. Model of Intel® CoFluent™ Technology for Big Data *(Intel® CoFluent™).*

Intel® CoFluent™ Technology for Big Data

Intel CoFluent is a customized modeling and simulation tool for analyzing the performance of big data clusters (see Figure 1). One of the ways Intel CoFluent delivers a high degree of accuracy in its simulations is by acquiring the execution time of workloads. Intel CoFluent acquires these execution times by mapping software behaviors to hardware components. This information can help both expert and less experienced developers optimize and better design big data clusters according to their business needs.

Basically, Intel CoFluent helps developers deploy clusters faster, optimize software parameters more accurately for their use cases, and scale with greater confidence by predicting cluster performance and behavior. Predictions no longer need to be a best guess that can be made only by a highly experienced developer.

With Intel CoFluent, optimization and other predictions can be based on:

- Highly accurate analyses of various nodes and disk configurations
- Optimizations of the software stack to more fully leverage hardware resources
- Conducting more accurate what-if analyses for big data clusters
- Exploring performance against the number of cluster users and cluster size

For these reasons, we chose Intel CoFluent as the modeling and simulation tool for our research.

Experiment environment

In this experiment, we ran TPCx-BB version 1.0 on a cluster of 1 master node and 7 slave nodes. (Table 1 provides an overview of the hardware components for the test cluster.) Each node is a Dell PowerEdge R730xd Rack Server* (R730xd* server), which is the new, thirteenth generation of Dell PowerEdge* servers. The R730xd servers offer an optimal balance of storage utilization, performance, and cost. These servers also offer an optional in-server hybrid storage configuration.

In this research project, we used an Apache Hadoop* software stack based on the Apache Cloudera Hadoop* distribution (CDH 5.12.0). We executed the 30 hive queries of TPCx-BB on an Apache Hadoop MapReduce* engine. Some queries used Apache Spark MLlib* and Apache Spark Mathout* libraries.

Table 1. Cluster settings for our big data experiment

Cluster component	Quantity	Description
Node count	8	1 master and 7 slaves
Processor	2	Intel® Xeon® processor E5-2660 v3 2.60GHz
Disk	12	Seagate Constellation* hard drive SATA 1TB 2 6Gbps 1TB 7.2K
DRAM memory	1	SK Hynix Ram Hynix* 16GB 2Rx4 PC4-2133P
Network	2	Intel® Ethernet Converged Network Adapter X520
Operating system	1	Red Hat, Inc., CentOS* 7.3
Software framework	1	Apache Hadoop* version CDH 5.12.0
Workload	1	TPCx-BB Express Benchmark (TPCx-BB)*[1]

Software parameter optimizations

Because of the scale of big data clusters, it is crucial that developers make the best use of a cluster's hardware resources. However, even for expert developers, it is extremely challenging to figure out the best parameter settings for an entire big data software stack.

Our Intel-Dell collaboration in research was focused on making it easier for developers to identify the optimizations they need. This could reduce development time. It could also allow developers to optimize many more parameters more accurately during the development cycle, to meet specific workload needs. With more workload-specific optimizations, businesses can better focus the offerings of their big data clusters. This includes using virtualization and use case profiles to dynamically adapt the cluster to different workloads based on user needs.

Challenges in optimizations

Consider the challenge of optimizing a big data workload. The operating system (OS), Oracle Java* virtual machine (JVM), and Hadoop software stacks have many parameters that can have a huge impact on performance.

First, the OS kernel includes parameters that configure Huge Page, Transparent Huge Page, flush intervals, and other kernel features. Then there is the JVM, which includes parameters that configure heap size; the ratio of old versus young generation parts of the heap; and the garbage collection (GC) algorithms. Also, the Hadoop software stack includes the largest number of parameters that are sensitive to the hardware performance of the system. These software stack parameters include settings for resource management, the execution engine, and various libraries.

It's common to gain a performance increase of several full factors just by optimizing software parameters in a big data cluster. However, it's still difficult even for highly experienced developers to figure out the best settings for all parameters in the workload.

One of the things that makes the process so challenging is that the best overall settings often change with different workload characteristics. For example, a CPU-intensive workload could require different optimal settings than an I/O-intensive workload. Optimal settings can also change with different hardware platforms. For example, a platform might be configured for CPU optimization, versus storage optimization, versus network optimization. In addition, parameters from different software layers can have tricky interactions. There is simply no general rule to follow that can determine the best parameter settings for a full software stack.

Modeling a workload with Intel CoFluent

Intel CoFluent is designed to help developers find the optimal software parameter settings faster, more easily, and more accurately for any given configuration. Since Intel CoFluent can simulate all software layers of a big data

ecosystem, it can not only analyze software parameter settings, but also identify the best settings for various configurations.

In our experiment, the software parameters could be divided into two types. One group of parameters consists of the 30 benchmark queries, all having the same setting. The other group consists of different queries, each having different settings.

In our experiment, we optimized values for parameters from the YARN, MapReduce, and Hive layers of the Hadoop software stack. Table 2 shows the values used for optimized settings for these parameters. Appendix A shows the values for the group of different queries that have different settings.

Establishing the accuracy of our model

We built our hardware model based on a known-good big data cluster used by the Dell Innovation Center,* located at the University of Pisa,* in Italy. The university provided us with the details of their hardware components, as well as the optimal big data settings which they had determined for that cluster.

To establish the accuracy of our model, we first built a hardware cluster (described previously in Table 1) similar to the University of Pisa's server system. We then simulated the cluster using Intel CoFluent. We compared the performance of the Intel CoFluent simulation against that of our hardware cluster. Our Intel CoFluent simulation achieved a performance accuracy of 96%, as compared to the performance of our physical hardware cluster.

We wanted to verify those performance results, for both our physical cluster and our simulations. To help with this, the Dell Innovation Center in Italy copied our optimizations and analyzed the resulting performance on their known-good physical cluster. They then provided our team with logs and feedback on the accuracy of our model.

Their results confirmed that our Intel CoFluent simulation delivered an accuracy of 96% when compared to performance of the university's known-good cluster. This accuracy impressed the university developer experts, as well as the Dell* developers.

Adjusting and testing the model

For our experiment, we first obtained a set of software parameter settings for our test configuration. These settings came from expert Intel developers based on their extensive experience in optimizing big data clusters. With their input, we adjusted our model, tested it, and achieved a TPCx-BB score of 85 (see Table 3).

We then used Intel CoFluent to simulate, analyze, and optimize all potential software parameter settings for our test configuration. This yielded a list of parameter settings which gave us a benchmark score of 112 (higher is better). That's a 32% increase compared to the expert-optimized settings. Table 3 lists the scores from the experts versus our simulation, using a 1TB workload input size.

Table 2. Simulation values for optimized settings of benchmark parameters

Parameter	Setting
yarn.nodemanager.resource.cpu-vcores	40
yarn.nodemanager.resource.memory-mb	160 GB
mapreduce.map.memory.mb	4 GB
mapreduce.reduce.memory.mb	4 GB
yarn.app.mapreduce.am.resource.mb	4GB
mapreduce.map.java.opts.max.heap	3072 MB
mapreduce.reduce.java.opts.max.heap	3027 MB
yarn.app.mapreduce.am.command-opts	3027 MB

Table 3. TPCx-BB scores of expert developers versus simulation results

Optimized settings	Benchmark score BBQpm@1000 (higher is better)
Optimized by expert developer	85
Optimized by Intel® CoFluent™ Technology for Big Data	112

Table 4. Summary of results by workload category

Performance improvement is sensitive to...	Query categories		
	Structured queries	Semi-structured queries	Unstructured queries
	1, 6, 7, 9, 11, 13 — 17, 20 — 26, 29	2, 3, 4, 5, 8, 12, 30	10, 18, 19, 27, 28
Scaling the processor	Average 20% decrease in execution time	Average 18% decrease in execution time	Average 23% decrease in execution time
Increasing network bandwidth	Small average 6% decrease in execution time	Minimal average 3% decrease in execution time	Minimal average 1% decrease in execution time
Scaling the number of nodes by a factor of 2	Average 47% decrease in execution time	Average 33% decrease in execution time	Average 12% decrease in execution time

Results

Once we had verified both our physical model and the accuracy of our simulations, we were ready to perform an analysis of optimizations chosen by human experts, versus options recommended by our simulation. Using both groups of settings, we analyzed the execution times for three types of upgrades or scaling:

- Upgrade the processor
- Upgrade the Ethernet
- Scale the number of nodes in the system

Based on data type, the TPCx-BB queries (workloads) can be grouped into three categories. Our results showed that specific groups or even specific queries were particularly sensitive to the biggest improvements, depending on the test component that was scaled.

POC results are summarized in Table 4. The data, query by query, are shown in figures 2, 3, and 4. In each of the graphs, the data is normalized to 1 for the baseline model, which was measured before any upgrade or scale analysis was performed.

Execution times for different processors

In our research, we changed usage scenarios (including input size and response time requests) in order to explore the challenge of scaling a cluster. In these scenarios, users typically need to scale their big data cluster to meet new requests. Our research shows that Intel CoFluent can provide highly effective scale-up and scale-out prediction analyses for big data clusters.

Figure 2 (next page) shows the results, as measured by TPCx-BB, of upgrading the processor.

For this test, we upgraded the processor from an Intel® Xeon® processor E5-2660 v3 to an Intel® Xeon® processor E5-2667 v4. The upgrade resulted in an average decrease in execution time of 20%.

As you can see in Figure 2, all 30 TPCx-BB queries showed similar reductions in execution time when the processor is upgraded. One query showed the greatest reduction in execution time, a 44% reduction.

Upgrading the processor to reduce execution times gives developers two options. They can run workloads on the existing cluster configuration with reduced power consumption. This can save energy and still make resources available for unexpected loads. Or, developers can recommend that cluster size be reduced while still providing competitive performance.

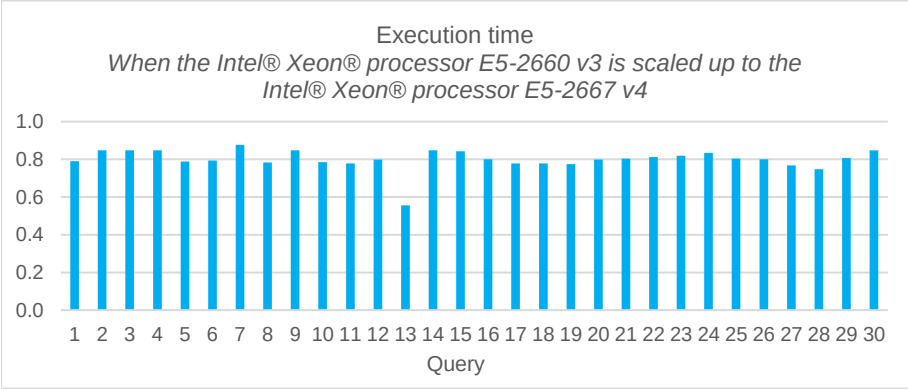

Figure 2. TPCx-BB analysis when the processor is upgraded from the Intel® Xeon® processor E5-2660 v3 to Intel® Xeon® processor E5-2667 v4. Data is normalized to 1 for the baseline configuration with the Intel Xeon processor E5-2660.

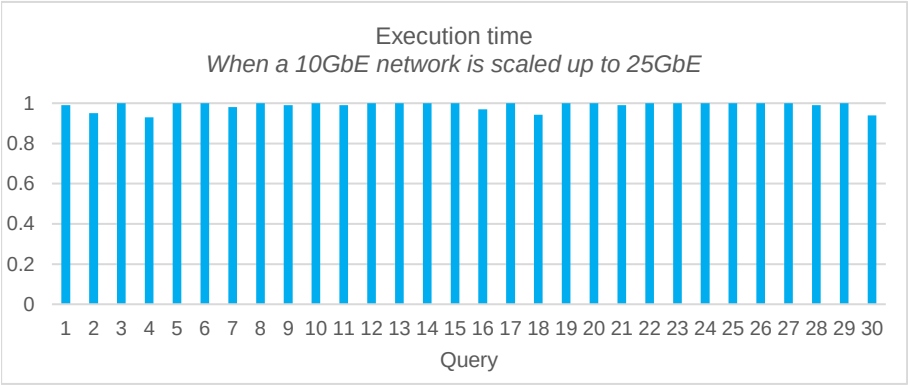

Figure 3. TPCx-BB results for execution times when the network is upgraded from 10 GbE to 25 GbE. Data is normalized to 1 for the baseline configuration with the 10 GbE network.

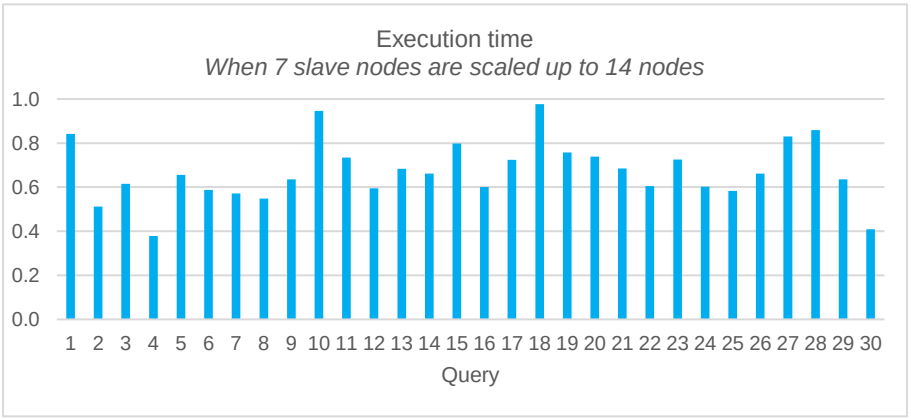

Figure 4. TPCx-BB results when nodes are scaled by a factor of two, from 7 to 14 slave nodes. Data is normalized to 1 for the baseline configuration with 7 slave nodes.

Execution times when the Ethernet is upgraded

Figure 3 shows how changes in network bandwidth affected our test configuration. As you can see in the figure, when upgrading the network bandwidth from 10 GbE to 25 GbE, execution time decreased an average of only 1%. Only a few queries (2, 4, 16, 18, and 30) with large-sized intermediate results achieved a larger decrease in execution time. (Intermediate results are the output of map tasks and the input of reduce tasks; and are transferred on the network from map to reduce.)

This information could be significant when identifying the best bandwidth parameters for a specific type of workload.

Execution times when nodes are scaled

Figure 4 shows the results of scaling the test configuration nodes by a factor of 2, from 7 slave nodes to 14 slave nodes. Scaling by a factor of 2 resulted in an average 33% decrease in execution time.

Based on our analysis, we believe that workloads similar to queries 2, 4, and 30 on large input tables will benefit more from the higher parallelism provided by additional nodes. We do not believe that workloads similar to queries 1, 10, 18, 27, and 28, on smaller input tables will be able to make full use of the resources made available by additional nodes.

Summary

For big data clusters, the values of thousands of configuration parameters can have a significant impact on performance. However, for convenience, developers often use default values. Choosing specific, optimal values for any given workload requires a developer with a high degree of experience and skill.

Our research project was a collaboration of Intel and Dell teams, via the Dell Innovation Center at the University of Pisa. Working together, our combined research team focused on how to use modeling and simulation — through the use of Intel CoFluent Technology for Big Data — to address the challenge of identifying optimal parameter values for big data clusters. We also used the TPCx-BB benchmark to help analyze the best way to scale big data systems for specific workloads.

Our results show that Intel CoFluent simulations can be an average of 32% more effective than the manual choices of expert developers. Such simulations can help experts significantly improve upon their hard-won software experience and optimize even more quickly for specific workloads, or simply optimize many more configuration parameters during a development cycle.

In this paper, we used a small cluster as a model, but Intel CoFluent is not limited by cluster size. Developers can use Intel CoFluent effectively to simulate and analyze medium- and large-sized clusters with more than 1000 nodes. For example, our experiment showed that big data workloads are sensitive to the number of nodes in a configuration. When scaling the system, queries with larger input sizes can see a much greater performance gain from having more cluster nodes.

Storage, network bandwidth, and other components can also create performance bottlenecks. Our research shows that big data workloads see little to no improvement when the Ethernet bandwidth is doubled, but they are definitely CPU-sensitive. To see better performance across a big data cluster, a priority could be to focus not on network bandwidth, but simply to upgrade the cluster's processors.

The results of our research show that, for the 30 workloads of the TPCx-BB benchmark, Intel CoFluent is a tool with a 96% accuracy when compared to physical clusters. With this kind of accuracy, Intel CoFluent can be an excellent tool to help developers address the significant challenge of identifying optimal parameters for big data clusters. With better optimizations, developers can choose between maximum performance or reducing the energy consumption required for specific workloads.

The results of our study also indicate that TPCx-BB is an effective standard for measuring CPU-sensitive workloads. Using TPCx-BB along with Intel CoFluent can help developers make full use of their expertise to tune big data clusters to make better use of resources. This can help reduce development time and improve the accuracy of early architecture designs; as well as reduce the total cost of ownership of big data clusters.

Appendix A
Optimized parameter settings for benchmark queries

In our experiment, the software parameters we optimized could be divided into two types. One type of parameter consisted of the 30 benchmark queries, all having the same setting. The other type consisted of different queries, each having different settings. This appendix shows the optimized values for parameters for different benchmark queries that had different settings.

```
#!/bin/bash

echo "**********set parameters for Query 01 ***************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set mapreduce.task.io.sort.mb=512;
set hive.exec.compress.intermediate=false;
set mapreduce.input.fileinputformat.split.maxsize=134217728;" >&
./engines/hive/queries/q01/engineLocalSettings.sql

echo "**********set parameters for Query 02 ***************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set mapreduce.output.fileoutputformat.compress=true;
set hive.exec.reducers.bytes.per.reducer=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.exec.compress.intermediate=true;" >& ./engines/hive/queries/q02/engineLocalSettings.sql

echo "**********set parameters for Query 03 ***************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.task.io.sort.mb=512;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set hive.exec.reducers.bytes.per.reducer=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.mapjoin.smalltable.filesize=400000000;" >& ./engines/hive/queries/q03/engineLocalSettings.sql

echo "**********set parameters for Query 04 ***************************"
echo -e "set hive.exec.reducers.max=280;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set hive.mapjoin.smalltable.filesize=400000000;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set mapreduce.output.fileoutputformat.compress=true;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.exec.compress.intermediate=true;" >& ./engines/hive/queries/q04/engineLocalSettings.sql

echo "**********set parameters for Query 05 ***************************"
echo -e "set hive.exec.reducers.max=280;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set hive.mapjoin.smalltable.filesize=400000000;
set mapreduce.output.fileoutputformat.compress=true;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set hive.exec.reducers.bytes.per.reducer=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
set hive.exec.compress.intermediate=true;" >& ./engines/hive/queries/q05/engineLocalSettings.sql
```

```
echo "**********set parameters for Query 06 ****************************"
echo -e "set hive.optimize.correlation=true;
set hive.exec.compress.intermediate=true;
set hive.exec.compress.output=false;
set hive.exec.reducers.max=280;
set hive.optimize.sampling.orderby=true;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.optimize.ppd=true;
set hive.optimize.ppd.storage=true;
set hive.ppd.recognizetransivity=true;
set hive.optimize.index.filter=true;
set hive.optimize.sampling.orderby.percent=0.2;" >& ./engines/hive/queries/q06/engineLocalSettings.sql

echo "**********set parameters for Query 07 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
" >& ./engines/hive/queries/q07/engineLocalSettings.sql

echo "**********set parameters for Query 08 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set hive.exec.reducers.bytes.per.reducer=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.exec.parallel=true;
set hive.exec.parallel.thread.number=8;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q08/engineLocalSettings.sql

echo "**********set parameters for Query 09 ****************************"
echo -e "set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
set hive.exec.compress.intermediate=false;
set hive.exec.mode.local.auto=true;
set hive.exec.mode.local.auto.inputbytes.max=1500000000;
set hive.mapjoin.smalltable.filesize=25000000;
set hive.exec.reducers.max=280;" >& ./engines/hive/queries/q09/engineLocalSettings.sql

echo "**********set parameters for Query 10 ****************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.input.fileinputformat.split.maxsize=4194304;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
set hive.exec.reducers.bytes.per.reducer=16777216;" >& ./engines/hive/queries/q10/engineLocalSettings.sql

echo "**********set parameters for Query 11 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q11/engineLocalSettings.sql

echo "**********set parameters for Query 12 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.exec.reducers.bytes.per.reducer=4194304;
set hive.mapjoin.smalltable.filesize=400000000;
```

```
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q12/engineLocalSettings.sql

echo "*********************set parameters for Query 13 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q13/engineLocalSettings.sql

echo "*********************set parameters for Query 14 ****************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
" >& ./engines/hive/queries/q14/engineLocalSettings.sql

echo "*********************set parameters for Query 15 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q15/engineLocalSettings.sql

echo "*********************set parameters for Query 16 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set mapreduce.output.fileoutputformat.compress=true;
set hive.exec.compress.intermediate=true;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
--set hive.groupby.skewindata=true;" >& ./engines/hive/queries/q16/engineLocalSettings.sql

echo "*********************set parameters for Query 17 ****************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.input.fileinputformat.split.maxsize=33554432;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q17/engineLocalSettings.sql

echo "*********************set parameters for Query 18 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.exec.reducers.bytes.per.reducer=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set mapreduce.input.fileinputformat.split.maxsize=33554432;" >&
./engines/hive/queries/q18/engineLocalSettings.sql

echo "*********************set parameters for Query 19 ****************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.task.io.sort.mb=512;
```

```
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
--set hive.optimize.skewjoin.compiletime=true;
set mapreduce.input.fileinputformat.split.maxsize=16777216;
set hive.exec.reducers.bytes.per.reducer=8388608;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q19/engineLocalSettings.sql

echo "********************set parameters for Query 20 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.exec.reducers.bytes.per.reducer=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set mapreduce.task.io.sort.factor=100;
set mapreduce.task.io.sort.mb=512;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set mapreduce.map.sort.spill.percent=0.99;
set mapreduce.input.fileinputformat.split.maxsize=67108864;" >&
./engines/hive/queries/q20/engineLocalSettings.sql

echo "********************set parameters for Query 21 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.exec.reducers.bytes.per.reducer=8388608;
set hive.optimize.correlation=true;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;" >& ./engines/hive/queries/q21/engineLocalSettings.sql

echo "********************set parameters for Query 22 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set mapreduce.input.fileinputformat.split.maxsize=67108864;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q22/engineLocalSettings.sql

echo "********************set parameters for Query 23 ****************************"
echo -e "set mapreduce.input.fileinputformat.split.maxsize=16777216;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.exec.reducers.bytes.per.reducer=67108864;
set hive.auto.convert.join.noconditionaltask.size=100000000;
set hive.exec.mode.local.auto=true;
set hive.exec.max.created.files=1000000;
set hive.exec.reducers.max=280;
set hive.exec.mode.local.auto.input.files.max=900;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;" >& ./engines/hive/queries/q23/engineLocalSettings.sql

echo "********************set parameters for Query 24 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q24/engineLocalSettings.sql

echo "********************set parameters for Query 25 ****************************"
echo -e "set hive.exec.reducers.max=280;
```

```
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set hive.exec.reducers.bytes.per.reducer=8388608;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q25/engineLocalSettings.sql

echo "********************set parameters for Query 26 ****************************"
echo -e "set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set hive.exec.reducers.bytes.per.reducer=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q26/engineLocalSettings.sql

echo "********************set parameters for Query 27 ****************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.input.fileinputformat.split.maxsize=8388608;
set mapreduce.job.reduce.slowstart.completedmaps=0.01;" >& ./engines/hive/queries/q27/engineLocalSettings.sql

echo "********************set parameters for Query 28 ****************************"
echo -e "set hive.exec.reducers.max=280;
set bigbench.hive.optimize.sampling.orderby=false;" >& ./engines/hive/queries/q28/engineLocalSettings.sql

echo "********************set parameters for Query 29 ****************************"
echo -e "set hive.exec.reducers.max=280;
set mapreduce.job.reduce.slowstart.completedmaps=0.5;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.exec.compress.intermediate=false;
set hive.exec.compress.output=false;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=134217728;

set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q29/engineLocalSettings.sql

echo "********************set parameters for Query 30 ****************************"
echo -e "set hive.exec.compress.intermediate=true;
set hive.exec.reducers.max=280;
set hive.mapjoin.smalltable.filesize=400000000;
set hive.optimize.correlation=true;
set hive.auto.convert.join.noconditionaltask=true;
set mapreduce.input.fileinputformat.split.maxsize=134217728;
set mapreduce.job.reduce.slowstart.completedmaps=0.9;
set hive.groupby.skewindata=false;
set hive.exec.parallel=true;
set hive.exec.parallel.thread.number=8;
set hive.auto.convert.join.noconditionaltask.size=100000000;" >&
./engines/hive/queries/q30/engineLocalSettings.sql
```

For more information about Intel CoFluent technology, visit cofluent.intel.com

For more information about the TPCx-BB benchmark, visit www.tpc.org/tpcx-bb/default.asp/

1 TPCx-BB source: http://www.tpc.org/tpcx-bb/default.asp/

2 BigBench: Towards an Industry Standard Benchmark for Big Data Analytics; Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, Hans-Arno Jacobsen; SIGMOD13, June 22, 2013, New York, New York, USA.

those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting **www.intel.com/design/literature.htm**.

Printed in USA.