Faster big data analytics and better responsiveness with IBM Cloud

An IBM Cloud configuration completed a big data analytics workload in less time and with greater throughput than an AWS solution

The cloud offers elasticity and flexibility to meet a range of demands for big data initiatives. Not all infrastructure-as-a-service (IaaS) providers are the same, though, and your organization requires performance and speed advantages to make big data initiatives successful.

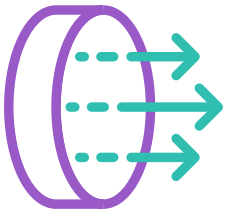
At Principled Technologies, we ran a Hadoop-based big data workload and a series of networking tests on two IaaS providers: IBM® Cloud and Amazon® Web Services (AWS®) Elastic Compute Cloud® (EC2®). The IBM Cloud solution clustered data points on virtual machines (VMs) faster and had higher throughput while working with big data than the AWS solution did. Compared to AWS, an IBM Cloud solution offers shorter wait times for the big data analytics that will drive your next marketing campaign, app, or operational improvements.

The IBM Cloud solution transferred data and sent pings between data centers in less time, too. Faster data delivery to and from data centers in the United States (US), United Kingdom (UK), and Japan means your global workforce could communicate, plan, and collaborate faster.

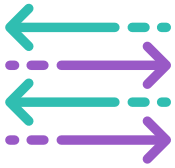
Your organization's IaaS provider affects your big data initiatives and trans-global communication. When it comes to performance and speed for big data, our findings show that an IBM Cloud solution can benefit your organization more than an AWS solution.

An IBM Cloud solution can help you find valuable insight from big data more quickly with...

28% less time clustering data



39% more clustering per second



up to 3x shorter data transfers

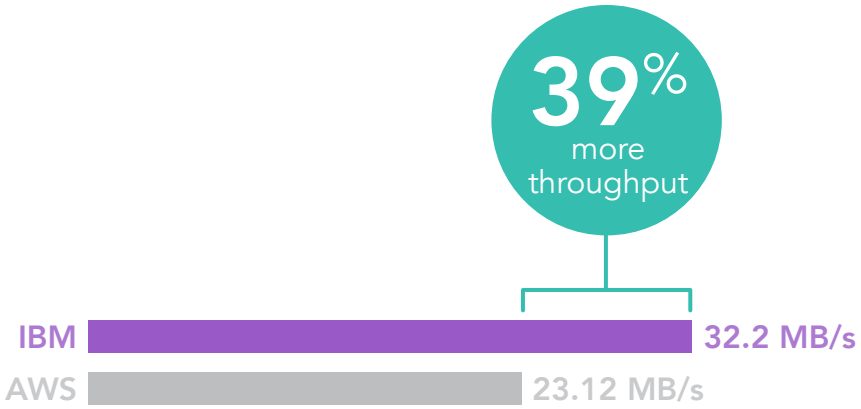
Choosing the cloud for big data initiatives brings flexibility and scalability

Big data analysis helps drive decision-making and improve areas such as customer service and product development by gathering, sorting, categorizing, and separating continuously growing data sets from a host of devices and users. These initiatives need infrastructure that allows data analysts, developers, and other IT staff to seamlessly scale their work to keep up with the growth of data. The cloud offers this kind of scalability and flexibility as well as these other advantages:

- There's little setup beyond choosing a configuration during the ordering process.
- IT admins have no additional on-site hardware to maintain.
- Organizations that can't afford or don't want additional on-premises computing resources don't have to incur hardware capital expenditure (CapEx) and operating expenditure (OpEx)—power, cooling, and management—just to get big data analytics.
- If an organization wants to run big data analytics for a short time or at limited intervals, they generally pay for only the time they use the cloud.

Work quickly with more data

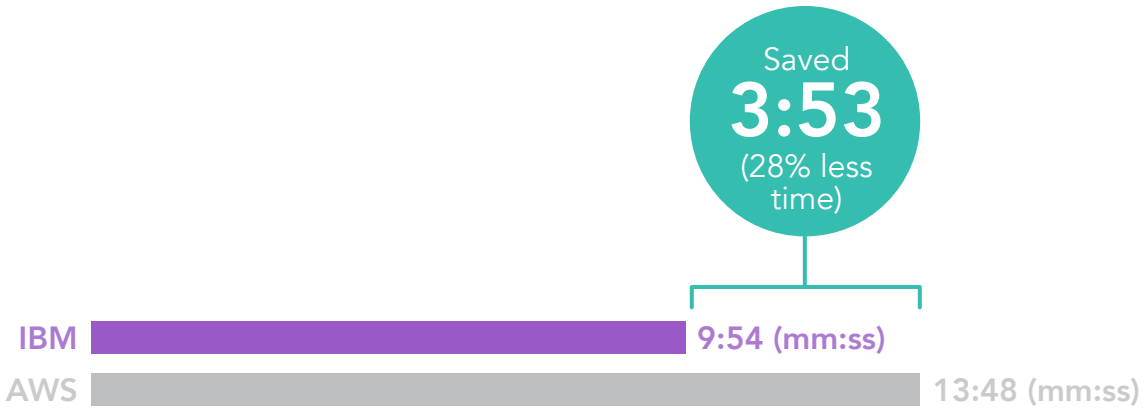
We ran the Kmeans workload from the HiBench suite of big data benchmarks. Kmeans is a machine learning benchmarking tool that clusters data points and reports throughput and duration of the workload. When we looked at the output from testing, the IBM Cloud solution had better throughput than our AWS configuration—a 39 percent increase (see Appendix B for more information on how we tested). When your virtualized solution delivers high throughput, you may need fewer VMs to handle your big data workloads, potentially reducing OpEx to get analytics and work with large data sets quickly.



Running analytics on more data per second allows analysts to work with larger data sets without significantly lengthening the duration of the workload. You can scale your IBM Cloud solution to handle greater big data workloads when the data sets you need to analyze continue to grow.

Cluster data more quickly for analysis

Whether you're looking to create effective marketing campaigns or trying to improve operational efficiency in your warehouse, the sooner a solution partitions data points into clusters, the sooner data analysts can see valuable insights and patterns. When clustering 18.7 GB of data for the Kmeans test, the IBM Cloud solution saved 3 minutes and 53 seconds compared to our AWS solution (28 percent less time). For more information on how we tested, see Appendix B.



Getting insights and patterns quickly, even with a small set of data, can mean the difference between turning one-time customers into returning shoppers and failing to capitalize on a revenue-generating demographic. Big data analytics is useful for many industries, not just retail. Even in healthcare, saving time can mean making the right diagnosis for a patient or cycling through another treatment. From an operational standpoint, speed in the cloud keeps your organization from piling on VMs just to finish a workload.

Let your workloads guide your choice of IBM Cloud service

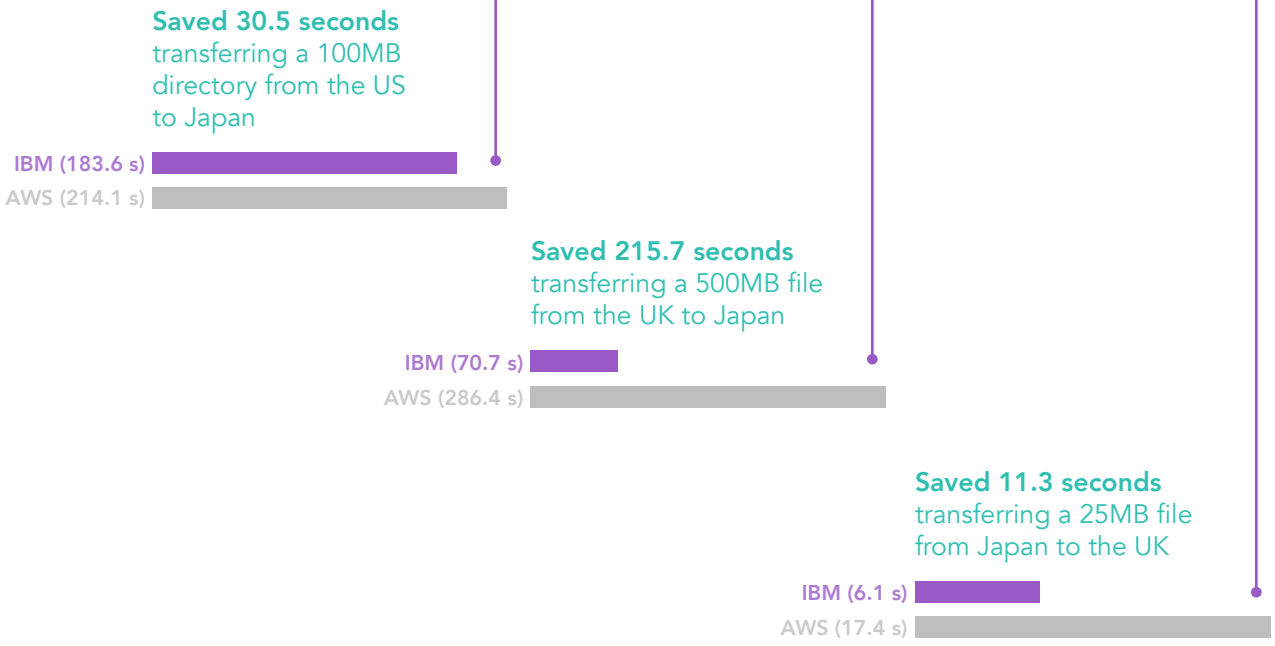
Some IaaS and cloud service providers offer limited services that developers and other IT staff must work around. The IBM solutions we tested were part of the more than 170 services from IBM Cloud, including bare-metal, public cloud, and container services. To balance cost with workload usage, IT and financial departments can choose between monthly and hourly pricing options. What's more, IBM Cloud has 60 locally owned and operated data centers in 19 countries around the world, so admins can choose to run applications and workloads in a specific location that ensures security and compliance.[1] In addition, IT admins choose compute resources from a range of Intel® Xeon® processors to best suite specific workloads. For more information, see https://www.ibm.com/cloud/.

Transfer data between geographically separate data centers more quickly

For businesses working on a global level, speedy transfers of data help managers and executives make informed, timely decisions. We transferred data between data centers in the US, UK, and Japan, and the IBM Cloud solution transferred data faster than AWS. When we transferred a 500MB file, we saw the biggest time savings for the IBM Cloud solution. From a UK data center to one in the US, transferring data with the IBM Cloud solution saved nearly two minutes compared to AWS. From the IBM Cloud UK data center to the one in Japan, transferring the 500MB file saved over three and a half minutes. Transferring the file from the US to Japan with the IBM Cloud solution saved over two and a half minutes compared to AWS. The figure below compares the times of the two solutions. For more information on how we tested, see Appendix B.

File and directory transfer (seconds)

25MB file						
	US to UK	**US to JP**	**UK to US**	**UK to JP**	**JP to US**	**JP to UK**
IBM Cloud	2.5	4.1	2.5	6.2	4.1	6.1
AWS	5.8	11.4	6.1	16.7	11.3	17.4
500MB file						
	US to UK	**US to JP**	**UK to US**	**UK to JP**	**JP to US**	**JP to UK**
IBM Cloud	24.7	46.2	24.5	70.7	46.2	70.9
AWS	104.0	208.6	143.3	286.4	199.5	281.4
100MB directory						
	US to UK	**US to JP**	**UK to US**	**UK to JP**	**JP to US**	**JP to UK**
IBM Cloud	99.5	183.6	98.9	280.1	183.2	281.7
AWS	106.2	214.1	112.1	292.8	188.6	293



Speedy data transfers help global organizations reduce the chance of operational bottlenecks and decreased productivity. For example, a financial institution with offices in the US, UK, and Japan would likely have to keep pace with markets and customer activity 24/7.

In addition to file transfers, we also sent pings between the data centers. The IBM Cloud solution won in five out of six ping tests. The largest difference between the two solutions came when we sent a ping between the US and Japan data centers—the IBM Cloud solution saved nearly 18 milliseconds. Faster pings indicate better responsiveness between data centers. The chart below compares the differences for pinging between the tested data centers of both solutions.

Ping test (milliseconds)						
	US to UK	**US to JP**	**UK to US**	**UK to JP**	**JP to US**	**JP to UK**
IBM Cloud	73.9	136.7	73.9	210.9	136.7	210.8
AWS	77.8	154.7	71.9	214.7	147.5	214.3

Conclusion

Big data analysis can drastically improve an organization. Being able to cluster and partition available data quickly helps analysts and decision-makers see in near real-time what users need, how effectively departments operate, or when is the best time to blast an email. We ran a Hadoop-based machine learning workload and networking tests on two solutions and found that an IBM Cloud solution clustered data more quickly, had greater throughput, and transferred data faster than an AWS solution. An IBM Cloud solution may be the IaaS platform your organization needs to reap the rewards of big data analytics and transfer data quickly between offices in the US, UK, and Japan.

1 "Bluemix is now IBM Cloud: Build confidently with 170+ services," January 18, 2018, https://www.ibm.com/blogs/bluemix/2017/10/bluemix-is-now-ibm-cloud/

On August 4, 2017, we finalized the hardware and software configurations we tested. Updates for current and recently released hardware and software appear often, so unavoidably these configurations may not represent the latest versions available when this report appears. For older systems, we chose configurations representative of typical purchases of those systems. We concluded hands-on testing on August 13, 2017.

Appendix A: System configuration information

Server configuration information	AWS server - Xen HVM domU
BIOS name and version	Xen 4.2
Operating system name and version/build number	Ubuntu 16.04.3 LTS
Date of last OS updates/patches applied	07/20/2017
Processor	
Number of processors	1
Vendor and model	Intel Xeon processor E5-2676 v3
Core count (per processor)	40
Core frequency (GHz)	2.40
Stepping	2
Memory module(s)	
Total memory in system (GB)	160
Number of memory modules	10
Size (GB)	16
Storage controller	
Vendor and model	Intel 82371SB PIIX3 IDE
Local storage	
Drive #1 size (GB)	100
Drive #2 size (GB)	500
Network adapter	
Vendor and model	Intel 82599 Ethernet Controller

Server configuration information	IBM Cloud server
Operating system name and version/build number	Ubuntu 16.04.3 LTS
Date of last OS updates/patches applied	07/20/2017
Processor	
Number of processors	1
Vendor and model	Intel Xeon processor E5-2683 v4
Core count (per processor)	32
Core frequency (GHz)	2.10
Stepping	1
Memory module(s)	
Total memory in system (GB)	128
Storage controller	
Vendor and model	Intel 82371SB PIIX3 IDE
Local storage	
Drive #1 size (GB)	350
Drive #2 size (GB)	500

Appendix B: How we tested

Designing the comparative scenario

For the comparative study portion of this analysis, we pitted the IBM Cloud configuration against the AWS configuration in measuring direct, real-world, CPU-intensive workload performance. We used the K-means Clustering tool from the Intel HiBench Hadoop benchmarking suite. We chose the tool to highlight the differences in mean computational throughput rather than synthetic (and hence less objectively useful) benchmarking tools that measure CPU performance on only a narrow task.

We configured the clouds to have similar performance to fairly assess the computational throughput from each. Owing to the limitations of vendor configuration offerings, we could not match configurations exactly. The following sections summarize the key commonalities and differences.

Common configuration

In each cloud, nodes served one of four roles: client (benchmarking machine), config (configuration management node), infrastructure (Hadoop name node, also called master nodes), or slave (Hadoop data nodes, i.e., system under test). We created VMs of four cores and 16 GiB of memory to satisfy the client, config, and infrastructure roles. For each VM, we allocated sufficient storage to support benchmarking and Hadoop infrastructure.

Creating the first VM in ESXi

All slave nodes used Intel Xeon processors E5-2600 v3 (virtualized).

IBM Cloud

- Each slave node had 32 threads (one CPU x 32 cores/CPU x one thread/core) and 128 GiB of RAM.
- The cloud solution had 128 threads across four nodes and 512 GiB of RAM total.

AWS

- Each slave node had 40 threads (one CPU x 40 cores/CPU x 1 thread/core) and 160 GiB of RAM.
- The cloud solution had 120 threads across three nodes and 480 GiB of RAM total.

Networking

All nodes were, as much as possible, co-located as closely as vendor provisioning options permitted and provisioned with the fastest networking possible. For the IBM Cloud solution, all nodes were located in Dallas 06 with VMs on 1Gbps networks. For the AWS solution, we specified a common placement group to co-locate the nodes and assigned each 10Gbps networking.

Storage

For client, config, and infrastructure roles, we provisioned the nodes with 25GB main drives of generic storage. The IBM Cloud solution used storage area network (SAN)-based drives. We configured the AWS solution with EBS (particularly General Purpose SSD (GP2)).

For slave nodes, we provided each a larger main/OS drive and a substantially larger drive to support Hadoop HDFS. The drives of virtual IBM Cloud nodes were SAN drives (100 GB for the OS, 500 GB for HDFS). We configured the slave nodes of the AWS solution with 100 GB for the OS and 500 GB for HDFS on 3,000 IOPS IO1 storage.

Methodology

Testing both solutions followed this general course:

1. Provision VMs with Linux Ubuntu 16.04 OS.
2. On infrastructure and slave nodes, install and configure Hadoop using Ambari.
3. On client nodes, install and configure Intel HiBench.
4. On client nodes, install and configure IBM LPCPU monitoring utility.
5. Prepare the HiBench workload.
6. Condition slave nodes for testing.
7. On slave nodes, start LPCPU monitoring.
8. Run HiBench Benchmark, and capture results.
9. On slave nodes, stop LPCPU monitoring, and capture results.
10. For each replication, complete steps 5 through 9.

Provisioning VMs with Linux Ubuntu 16.04

Each vendor's mechanism of provisioning differed slightly but followed the same general pattern:

1. Log into the web portal.
2. Click Devices or Instances.
3. Select appropriate values for the following:
   - Quantity
   - Region or Placement Group
   - Operating System
   - CPU
   - Memory
   - Networking
   - Disk
4. Confirm the reservation, and wait for VM to start.
5. Using SSH, log into the VM, and configure any software prerequisites to support Hadoop, Ambari, HiBench, LPCPU, or any configuration management software.
6. For each host in the cluster, perform the following steps:
   a. Connect to the host <XXX>:
      ```
      > ssh root@XXX
      ```
   b. Create an RSA keypair for the root user:
      ```
      > ssh-keygen -t rsa -b 8192 -N "" -f ~/.ssh/id_rsa
      ```
   c. For each other host YYY in the cluster, copy root@XXX's SSH key to YYY:
      ```
      > ssh-copy-id -i ~/.ssh/id_rsa.pub root@YYY
      ```
   d. Repeat setp 6c for each host in the cluster.
7. Repeat step 6 for each host in the cluster.

Installing and configuring Hadoop

1. On all nodes, install Ambari REPO and key:
   ```
   > wget -O /etc/apt/sources.list.d/ambari.list http://public-repo-1.hortonworks.com/ambari/
   ubuntu16/2.x/updates/2.5.1.0/ambari.list
   > apt-key adv --recv-keys --keyserver keyserver.ubuntu.com B9733A7A07513CAD
   ```
2. Update package cache, and apply upgrades:
   ```
   > apt-get update -y
   > apt-get upgrade -y
   ```
3. On the config node, install Ambari Server:
   ```
   > apt-get install -y ambari-server
   > ambari-server setup -j PATH_TO_JAVA_HOME -s
   > ambari-server restart
   ```
4. On the other nodes, install Ambari Agent:
   ```
   > apt-get install -y ambari-agent
   > nano /etc/ambari-agent/conf/ambari-agent.ini
      set 'hostname' to the IP/FQDN of the 'config' node
   > ambari-agent restart
   ```
5. On the config node, compute or set the following emboldened variables as indicated:
   - Number of slave nodes in the cluster:
     NUMBER_OF_SLAVES = **X**
   - Minimum amount of memory (MiB) on any node:
     MIN_MEM = **131072**
   - MB of memory to reserve for OS (e.g. 28 GB):
     OS_RESERVE_MB = **31072**
   - Number of threads per node :
     NUM_CORES = **num_cpus * cores/cpu * threads/core**
   - Compute the number of cores to allow:
     YARN_NUM_CORES = **NUM_CORES - 2**
   - Allowable Yarn memory:
     YARN_MEMORY_MB = **MIN_MEM - OS_RESERVE_MB = 100000**

- Yarn memory allocator:
  YARN_MIN_ALLOC_MB = **8192**
  YARN_MAX_ALLOC_MB = **32768**
  YARN_INCREMENT_MB = **512**
- Yarn vcpu allocator:
  YARN_MIN_CORES = **1**
  YARN_MAX_CORES = **30**
  YARN_INCREMENT_CORES = **1**
- MapReduce memory:
  MAPRED_MAP_MEMORY_MB = **4096**
  MAPRED_REDUCE_MEMORY_MB = **8192**

a. Create a "hostmap" that maps hosts to Hadoop roles in Ambari:

```
> nano ~/ambari-hostmap.json
 {
   "blueprint" :        "my_hadoop_cloud",
   "default_password" : "top-secret",
   "host_groups" :[
     { "name" : "master1", "hosts" : [ { "fqdn" : "master1.my_cloud.org" } ] },
     { "name" : "master2", "hosts" : [ { "fqdn" : "master2.my_cloud.org" } ] },
     { "name" : "master3", "hosts" : [ { "fqdn" : "master3.my_cloud.org" } ] },
     { "name" : "clients", "hosts" : [ { "fqdn" : "client.my_cloud.org"  } ] },
     { "name" : "slaves",
       "hosts" : [
         { "fqdn" : "slave1.my_cloud.org" },
         { "fqdn" : "slave2.my_cloud.org" },
         { "fqdn" : "slave3.my_cloud.org" },
         ...
         { "fqdn" : "slaveN.my_cloud.org" }
       ]
     }
   ]
 }
```

b. Create an Ambari blueprint:

```
> nano ~/ambari-blueprint.json

   "configurations" : [
     { "hive-site" : {
         "javax.jdo.option.ConnectionUserName" : "$HIVE_USER",
         "javax.jdo.option.ConnectionPassword" : "$HIVE_PASS" } },
     { "hdfs-site" : {
           "dfs.datanode.data.dir" : "$DISK_1_MOUNTPOINT, $DISK_2_MOUNTPOINT, ... $DISK_N_
MOUNTPOINT" } } },
     { "yarn-site" : {
         "properties" : {
           "yarn.nodemanager.resource.cpu-vcores"       : $YARN_NUM_CORES,
           "yarn.nodemanager.resource.memory-mb"        : $YARN_MEMORY_MB,
           "yarn.scheduler.minimum-allocation-mb"       : $YARN_MIN_ALLOC_MB,
           "yarn.scheduler.maximum-allocation-mb"       : $YARN_MAX_ALLOC_MB,
           "yarn.scheduler.increment-allocation-mb"     : $YARN_INCREMENT_MB,
           "yarn.scheduler.minimum-allocation-vcores"   : $YARN_MIN_CORES,
           "yarn.scheduler.maximum-allocation-vcores"   : $YARN_MAX_CORES,
           "yarn.scheduler.increment-allocation-vcores" : $YARN_INCREMENT_CORES } } },
     { "mapred-site" : {
         "properties" : {
           "mapreduce.map.memory.mb"      : $MAPRED_MAP_MEMORY_MB,
           "mapreduce.reduce.memory.mb"   : $MAPRED_REDUCE_MEMORY_MB } } }
   ],
   "host_groups" : [
     { "components" : [
         { "name" : "SECONDARY_NAMENODE"},
         { "name" : "HST_AGENT"},
         { "name" : "SLIDER"},
         { "name" : "SPARK_CLIENT"},
         { "name" : "HDFS_CLIENT"},
         { "name" : "ZOOKEEPER_SERVER"},
```

```
      { "name" : "HISTORYSERVER"},
      { "name" : "METRICS_MONITOR"},
      { "name" : "TEZ_CLIENT"},
      { "name" : "APP_TIMELINE_SERVER"},
      { "name" : "RESOURCEMANAGER"}
    ],
    "configurations" : [ ],
    "name" : "master2",
    "cardinality" : "1" },
  { "components" : [
      { "name" : "SPARK_CLIENT" },
      { "name" : "YARN_CLIENT" },
      { "name" : "HDFS_CLIENT" },
      { "name" : "HST_SERVER" },
      { "name" : "STORM_UI_SERVER" },
      { "name" : "METRICS_MONITOR" },
      { "name" : "INFRA_SOLR_CLIENT" },
      { "name" : "NAMENODE" },
      { "name" : "TEZ_CLIENT" },
      { "name" : "ZEPPELIN_MASTER" },
      { "name" : "NIMBUS" },
      { "name" : "KNOX_GATEWAY" },
      { "name" : "SPARK2_JOBHISTORYSERVER" },
      { "name" : "ACTIVITY_ANALYZER" },
      { "name" : "KAFKA_BROKER" },
      { "name" : "HST_AGENT" },
      { "name" : "MAPREDUCE2_CLIENT" },
      { "name" : "ZOOKEEPER_SERVER" },
      { "name" : "INFRA_SOLR" },
      { "name" : "SPARK_JOBHISTORYSERVER" },
      { "name" : "DRPC_SERVER" },
      { "name" : "METRICS_GRAFANA" }
    ],
    "configurations" : [ ],
    "name" : "master1",
    "cardinality" : "1" },
  { "components" : [
      {"name" : "MAHOUT" },
      { "name" : "SPARK_CLIENT" },
      { "name" : "YARN_CLIENT" },
      { "name" : "HDFS_CLIENT" },
      { "name" : "SPARK2_CLIENT" },
      { "name" : "METRICS_MONITOR" },
      { "name" : "INFRA_SOLR_CLIENT" },
      { "name" : "TEZ_CLIENT" },
      { "name" : "ZOOKEEPER_CLIENT" },
      { "name" : "HCAT" },
      { "name" : "PIG" },
      { "name" : "HST_AGENT" },
      { "name" : "MAPREDUCE2_CLIENT" },
      { "name" : "SLIDER" },
      { "name" : "HIVE_CLIENT" }
    ],
    "configurations" : [ ],
    "name" : "clients",
    "cardinality" : "1"
  },
  { "components" : [
      { "name" : "YARN_CLIENT" },
      { "name" : "HDFS_CLIENT" },
      { "name" : "HIVE_SERVER" },
      { "name" : "MYSQL_SERVER" },
      { "name" : "METRICS_MONITOR" },
      { "name" : "HIVE_METASTORE" },
      { "name" : "TEZ_CLIENT" },
      { "name" : "ZOOKEEPER_CLIENT" },
      { "name" : "PIG" },
```

```
            { "name" : "WEBHCAT_SERVER" },
            { "name" : "HST_AGENT" },
            { "name" : "MAPREDUCE2_CLIENT" },
            { "name" : "ZOOKEEPER_SERVER" },
            { "name" : "HIVE_CLIENT" },
            { "name" : "METRICS_COLLECTOR" }
          ],
          "configurations" : [ ],
          "name" : "master3",
          "cardinality" : "1" },
        { "components" : [
            { "name" : "NODEMANAGER" },
            { "name" : "HST_AGENT" },
            { "name" : "DATANODE" },
            { "name" : "METRICS_MONITOR" },
            { "name" : "SUPERVISOR" }
          ],
          "configurations" : [ ],
          "name" : "slaves",
          "cardinality" : $NUMBER_OF_SLAVES }
      ],
      "settings" : [
        { "recovery_settings" : [ { "recovery_enabled" : "true" } ] },
        { "service_settings" : [
            { "name" : "HIVE",           "credential_store_enabled" : "true" },
            { "name" : "AMBARI_METRICS", "recovery_enabled" : "true" }
          ] },
        { "component_settings" : [
            { "name" : "METRICS_COLLECTOR", "recovery_enabled" : "true" }
          ] }
      ],
      "Blueprints" : {
        "blueprint_name": "my_hadoop_cloud",
        "stack_name"    : "HDP",
        "stack_version" : "2.6"
      }
    }
```

c. Register the blueprint:

```
> BLUEPRINT=`cat ~/ambari-hostmap.json`
> AMBARI_SERVER=ambari.my_cloud.org
> BLUEPRINT_NAME=my_hadoop_cloud
> URL="http://${AMBARI_SERVER}:8080/api/v1/blueprints/${BLUEPRINT_NAME}?validate_topology=false"
> curl -H "X-Requested-By: ambari-script" -d "$BLUEPRINT" -X POST -u admin:admin $URL
```

Note: you can also perform a **GET CURL** request to **http://${AMBARI_SERVER}:8080/api/v1/blueprints/${BLUEPRINT_NAME}** to confirm you've registered the blueprint successfully.

d. Register the cluster (host map)

```
> HOSTMAP=`cat ~/ambari-blueprint.json`
> AMBARI_SERVER=ambari.my_cloud.org
> CLUSTER_NAME=my_hadoop_cloud
> URL="http://${AMBARI_SERVER}:8080/api/v1/clusters/${CLUSTER_NAME}?validate_topology=false"
> curl -H "X-Requested-By: ambari-script" -d "$HOSTMAP" -X POST -u admin:admin $URL
```

6. Using a browser, navigate to **AMBARI_SERVER:8080**, log in, and monitor operation status until cluster deployment completes.

   Note: you can also perform a **GET CURL** request to **http://${AMBARI_SERVER}:8080/api/v1/clusters/${CLUSTER_NAME}/requests/1** to monitor the status of the deployment.

Installing HiBench

On client nodes, perform the following steps:

1. Install prerequisites:

```
> apt-get install -y maven git python-numpy libblas-common libblas-dev \
libblas3 liblapack-dev liblapack3 liblapacke \
liblapacke-dev bc
```

2. Clone HiBench:

```
> mkdir ~/HiBench
> cd ~/HiBench
> git clone https://github.com/intel-hadoop/HiBench.git .
```

3. Build HiBench:

```
> cd ~/HiBench
> mvn -Dspark=2.1 -Dscala=2.11 clean package
```

4. Disable SSH StrictHostKeyChecking:

```
> nano ~/.ssh/config

 Host *
    StrictHostKeyChecking no

 [...]
```

Note: Step 4 allows you to generate monitoring/utilization reports.

Installing LPCPU

On slave nodes, perform the following steps:

1. Install prerequisites:

```
> apt-get install -y sysstat
```

2. Download LPCPU tarball:

```
> wget -o ~/lpcpu.tar.bz2 "https://www.ibm.com/developerworks/community/wikis/form/anonymous/
api/wiki/26579cc3-66fe-42b8-baf9-1fcc88445848/page/4a7d2717-05c8-4b78-886e-5e4f9fdf07c1/
attachment/7018590b-7fbe-4852-8d44-79af2d83fffa/media/lpcpu.tar.bz2"
```

3. Extract LPCPU:

```
> cd ~
> tar -xjvf lpcpu.tar.bz2
```

Preparing HiBench

1. Compute or note the following emboldened quantities:

- The user name for HDFS
  HDFS_USER = **XXX**
- The fully qualified domain name of the first master node
  MASTER_1_FQDN = **master1.my_cloud.org**
- The fully qualified domain names of all master nodes, separated with spaces
  MASTER_FQDNS = **master1.my_cloud.org master2.my_cloud.org master3.my_cloud.org**
- The fully qualified domain names of all slave nodes, separated with spaces
  SLAVE_FQDNS = **slave1.my_cloud.org slave2.my_cloud.org slave3.my_cloud.org**
- The size of the workload tiny, huge, gigantic, bigdata, etc.
  HIBENCH_SCALE = **gigantic**
- Slave count
  S = **Total number of slave nodes**
- Thread count
  C = **Sum of the number of CPUs/threads across all slaves**
- Minimum memory on any slave node, in GiB
  U = **128**
- Fraction of memory to occupy, 0.9 (90 percent) is a good number
  W: **0**
- Number of masters
  M: **3**

- Spark number of cores (5 is reasonable)
  EXECUTOR_CORES = **5**
- Spark executor num
  EXECUTOR_NUM = **floor((C-S) / EXECUTOR_CORES)**
- Spark memory
  EXECUTOR_MEMORY = **floor((U*S) / EXECUTOR_NUM)**
- Spark memory
  DRIVER_MEMORY = **floor((U*S) / EXECUTOR_NUM)**
- Spark map parallelism
  MAP_PARALLELISM = **C-S**
- Spark reduce parallelism
  SHUFFLE_PARALLELISM = **C-S**

2. Set HiBench Hadoop Config:

```
> cp ~/HiBench/conf/hadoop.conf.template ~/HiBench/conf/hadoop.conf
> nano ~/HiBench/conf/hadoop.conf

 [...]

 hibench.hadoop.home /usr/hdp/current/hadoop-client
 hibench.hdfs.master hdfs://${MASTER_1_FQDN}:8020/user/${HDFS_USER}"

 [...]
```

3. Set HiBench Spark Config:

```
> cp ~/HiBench/conf/spark.conf.template ~/HiBench/conf/spark.conf
> nano ~/HiBench/conf/spark.conf

 [...]

 hibench.spark.home             /usr/hdp/current/spark2-client
 hibench.yarn.executor.num      ${EXECUTOR_NUM}
 hibench.yarn.executor.cores    ${EXECUTOR_CORES}
 spark.executor.memory          ${EXECUTOR_MEMORY}
 spark.driver.memory            ${DRIVER_MEMORY}

 [...]
```

4. Set HiBench Spark Config:

```
> cp ~/HiBench/conf/hibench.conf ~/HiBench/conf/hibench.conf.orig
> nano ~/HiBench/conf/hibench.conf

 [...]
 hibench.scale.profile                ${HIBENCH_SCALE}
 hibench.streambench.kafka.home        /usr/hdp/current/kafka-broker
 hibench.default.map.parallelism       ${MAP_PARALLELISM}
 hibench.default.shuffle.parallelism  ${SHUFFLE_PARALLELISM}
 hibench.masters.hostnames             '${MASTER_FQDNS}'
 hibench.slaves.hostnames              '${SLAVE_FQDNS}'

 [...]
```

5. Initialize storage:
   a. If storage has previously initialized, clear it:

```
> sudo -u hdfs hdfs dfs -rmr /HiBench
> sudo -u hdfs hdfs dfs -rmr /user/${HDFS_USER}
> sudo -u hdfs hdfs dfs -expunge
> sudo -u hdfs hdfs dfs -rmr '/user/*/.Trash'
```

   b. Create required directories:

```
> sudo -u hdfs hdfs dfs -mkdir /HiBench
> sudo -u hdfs hdfs dfs -chown -R ${HDFS_USER}:hadoop /HiBench
> sudo -u hdfs hdfs dfs -mkdir /user/${HDFS_USER}
> sudo -u hdfs hdfs dfs -chown ${HDFS_USER} /user/${HDFS_USER}
```

6. Run preparation script:

```
> cd ~/HiBench
> ~/HiBench/bin/workloads/ml/kmeans/prepare/prepare.sh
```

Conditioning the slave nodes

On all slave nodes, perform the following steps:

1. Disable swap:

```
> swapoff -a
```

2. Re-enable swap:

```
> swapon -a
```

3. Flush page cache:

```
> [ -f /proc/sys/vm/drop_caches ] && echo 3 > /proc/sys/vm/drop_caches
```

Note: You could do this on all nodes, but we targeted only slave nodes in our testing.

Starting LPCPU monitoring

On all slave nodes, perform the following steps:

1. Start LPCPU monitoring:

```
> cd ~/lpcpu
> ./lpcpu.sh duration=99999999 output_dir=output interval=${INTERVAL} dir_name=data
```

Note: leave the SSH session open and LPCPU running (or wrap in start-stop-daemon, etc. to run in background) for the duration of the test.

Running HiBench K-means benchmark

On the client node, perform the following steps:

1. Run the benchmark:

```
> cd ~/HiBench
> ./bin/workloads/ml/kmeans/hadoop/run.sh
```

2. Collect the contents of ~/HiBench/report.

Stopping LPCPU monitoring

On all slave nodes, perform the following steps:

1. To interrupt the lpcpu.sh in the SSH session, type `[CTRL] + C`.
2. Wait for the tarball to be created (output/data.tar.bz2).
3. Extract the tarball, and run the postprocess.sh script.
4. Collect the contents of output/data directory.

Measuring bandwidth

We measured network performance between VMs in one of three regions (United States East Coast, United Kingdom, and Japan) to VMs in another of the three regions to understand network latency and throughput performance using TCP-based ping and secure copy tools. We tested these aspects of both services using paping to perform the tests.

We performed the TCP-ping tests by connecting to a web server running on the target cloud environment, and using the ping utility to measure the average latency to port 80 for 30 seconds, and took the median result of three tests.

We tested file copy performance using scp, and timed the duration of the transfer of 25MB and 500MB files, as well as a folder containing 100MB of small files.

Cloud instance configuration

Three VMs per service, U.S. East, Western Europe, and Asia Pacific (Japan), configured as follows:

Regions

- AWS: N. Virginia, Ireland, Asia Pacific (Tokyo)
- IBM Cloud: Washington DC, London, Tokyo

VM instance types

- All VMs run Ubuntu 14.

VM configurations

- AWS EC2: "m4.xlarge" general purpose image with 4 vCPUs, 16GB RAM, EBS storage, "High" network performance
- IBM Cloud: compute virtual server configured for 4x2.0GHz cores, 16GB RAM, 1 Gbps public & private network uplinks

This project was commissioned by IBM.

Facts matter.®