利用 IBM Cloud 加速大数据分析并改善响应能力

相比 AWS 解决方案，IBM Cloud 配置能够以更高吞吐量、用更少时间完成大数据分析。

这款云提供弹性和灵活性，以满足大数据计划的一系列要求。不过，并非所有基础架构即服务 (IaaS) 提供商都一样，您的组织需要性能和速度优势，以实现成功的大数据计划。

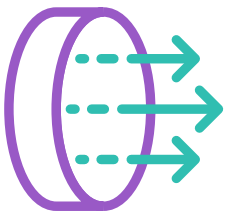
在 Principled Technologies，我们在以下两个 IaaS 提供商上运行了基于 Hadoop 的大数据工作负载和一系列的网络测试：IBM® Cloud 和 Amazon® Web Services (AWS®) Elastic Compute Cloud® (EC2®). 在处理大数据时，IBM Cloud 解决方案能够比 AWS 解决方案更快地在虚拟机 (VM) 上群集数据点，并具备更高的吞吐量。相比 AWS，IBM Cloud 解决方案为支持您的下次营销活动、应用或运营改进的大数据分析提供更短等待时间。

IBM Cloud 解决方案还能用更短时间在数据中心之间传输数据和发送 ping。更快地向/从位于美国 (US)、英国 (UK)和日本的数据中心交付数据，意味着您的全球员工队伍能够更快地进行沟通、规划和协作。

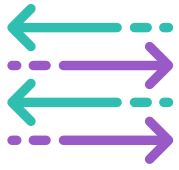
您组织的 IaaS 提供商影响您的大数据计划和全球沟通。在大数据性能和速度方面，我们的调查结果显示，IBM Cloud 解决方案为您组织带来的优势超过 AWS 解决方案。

凭借以下优势，IBM Cloud 解决方案可以帮助您 更快地从大数据 获取宝贵的洞察...

少 28% 的数据群集时间



高 39% 的每秒群集速度



多达 3 倍的数据传输速度

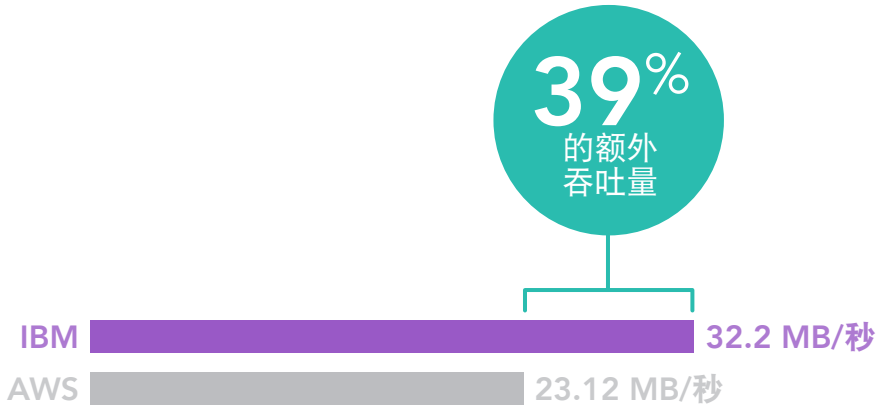
为大数据计划选择云，将带来灵活性和可扩展性

通过收集、整理、分类和分离来自众多设备和用户的持续增长的数据，大数据分析有助于推动决策制定，并改善客户服务和产品开发等领域。这些计划需要合适的基础架构，以支持数据分析师、开发者和其他 IT 员工无缝地扩展他们的工作，跟上数据增长步伐。这款云提供这种可扩展性和灵活性，以及以下众多优势：

- 除了在订购过程中选择配置，基本上无需设置。
- IT 管理员没有要维护的额外现场硬件。
- 如果组织负担不起或不需要额外的组织内部计算资源，则不必承受资本支出 (CapEx) 和运营支出 (OpEx)（包库电源、散热和管理），只需获得大数据分析。
- 如果组织希望在短时间内或以有限间隔运行大数据分析，他们通常仅为他们使用的时间付费。

更快地处理更多数据

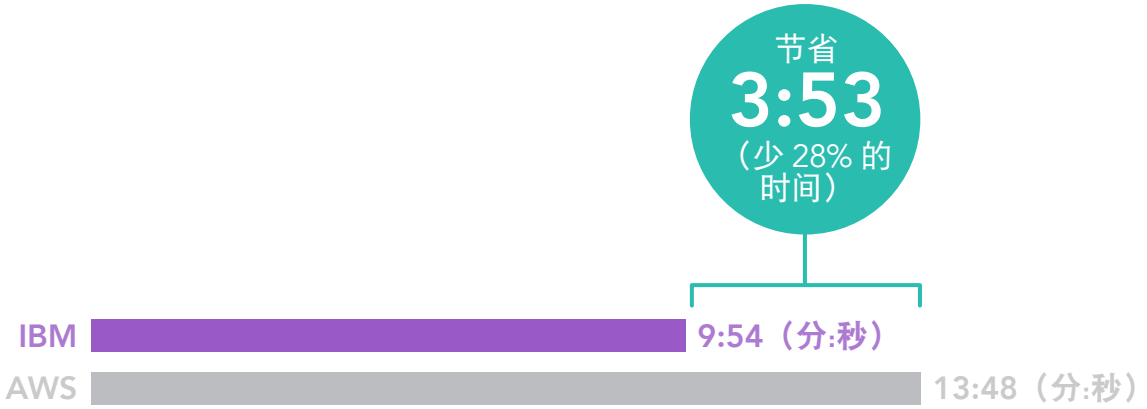
我们从 HiBench 大数据基准测套件运行 Kmeans 工作负载。Kmeans 是一款机器学习基准测试工具，用于群集数据点并报告工作负载吞吐量和持续时间。 测试的输出显示，IBM Cloud 解决方案的吞吐量高于我们的 AWS 配置，高出 39%（参见 附录 B，了解关于我们如何测试的更多信息）。当您的虚拟化解决方案提供高吞吐量时，您可能需要更少的虚拟机处理您的大数据工作负载，有望减少获得分析所需的运营支出，并快速处理大量数据集。



通过每秒对更多数据运行分析，分析师可以处理更大数据集，而不会大幅延长工作负载的持续时间。当需要分析的数据集不断增长时，您可以扩展您的 IBM Cloud 解决方案，以处理更多的大数据工作负载。

为分析更快地群集数据

无论您是考虑设计有效的营销活动，还是努力改善仓库的运营效率，解决方案越早将数据点划分到集群中，数据分析师就能越早看到宝贵的洞察和模式。 在为 Kmeans 测试群集 18.7 GB 数据时，IBM Cloud 比我们的 AWS 解决方案节省了 3 分 53 秒（少 28% 的时间）。了解关于我们如何测试的更多信息，参见 附录 B.



即便是对于小型数据集，快速获取洞察和模式的能力可能决定，您是将一次性客户转变为回头客，还是无法充分发掘可创收的人口群体。大数据分析对于许多行业都有用，并非仅限零售业。即便是在医疗保健业，节省时间可能决定您是为患者作出正确的诊断，还是循环使用另一套治疗方案。从运营的角度看，云中的速度让您的组织可以避免单纯为了完成工作负载而堆积虚拟机。

让您的工作负载指导您的 IBM Cloud 服务选择

一些 IaaS 和云服务提供商提供有限的服务，开发者和其他 IT 员工必须对此找到权变举措。 我们测试的 IBM 解决方案是 IBM Cloud 中 170 多项服务的一部分，其中包括裸机、公共云和容器服务。为了平衡成本与工作负载应用，IT 和财务部门可以在按月和按小时计费选项之间做出选择。此外，IBM Cloud 在全球 19 个国家或地区拥有 60 个本地拥有和运营的数据中心，因此管理员可以选择在可确保安全性与合规性的特定位置运行应用与工作负载。[1] 此外，IT 管理员可以选择来自一系列 Intel® Xeon® 处理器的计算资源，以适合特定工作负载。 如需更多信息，请参阅 https://www.ibm.com/cloud/.

更快地在两个地理上分离的数据中心之间传输数据

对于在全球开展业务的企业，快速数据传输有助于经理和高管及时作出明智的决策。我们在位于美国、英国和日本的数据中心之间传输数据，IBM Cloud 解决方案传输数据的速度快于 AWS。 当我们传输一个 500MB 的文件时，我看到 IBM Cloud 解决方案实现了最大的时间节省。 从位于英国的数据中心到位于美国的数据中心，利用 IBM Cloud 解决方案传输数据比 AWS 节省近两分钟。 从英国的数据中心到日本的数据中心，IBM Cloud 传输 500MB 文件节省了超过 3.5 分钟。 利用 IBM Cloud 解决方案从美国向日本传输文件比 AWS 节省超过 2.5 分钟。 下图显示了这两款解决方案的时间比较情况。 了解关于我们如何测试的更多信息，参见 附录 B.

文件和目录传输（秒）

25MB 文件

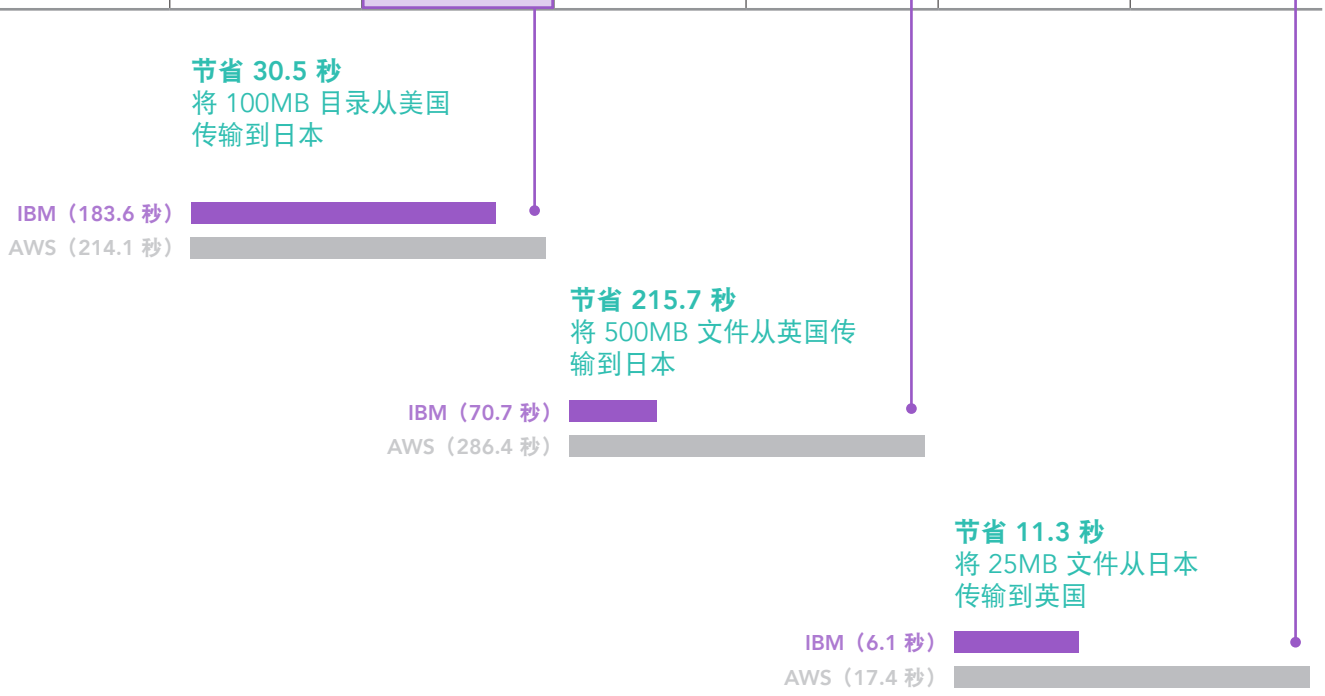
	美国到英国	美国到日本	英国到美国	英国到日本	日本到美国	日本到英国
IBM Cloud	2.5	4.1	2.5	6.2	4.1	6.1
AWS	5.8	11.4	6.1	16.7	11.3	17.4

500MB 文件

	美国到英国	美国到日本	英国到美国	英国到日本	日本到美国	日本到英国
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 目录

	美国到英国	美国到日本	英国到美国	英国到日本	日本到美国	日本到英国
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



快速数据传输有助于全球性组织减少出现运营瓶颈和低下生产效率的概率。 例如，一家在美国、英国和日本设有办公室的金融机构，可能希望全天候追踪市场和客户活动。

除了文件传输，我们还在数据中心之间发送 ping。 IBM Cloud 解决方案在六项 ping 测试中赢得五项。 当我们在美国和日本两地的数据中心之间发送 ping 时，两款解决方案出现最大差异，IBM Cloud 解决方案节省了近 18 毫秒。更快的 ping 表明两个数据中心之间的响应能力更强。 下图比较了两款解决方案在受测试数据中心之间的 ping 差异。

ping 测试（毫秒）

	美国到英国	美国到日本	英国到美国	英国到日本	日本到美国	日本到英国
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

结论

大数据分析可以大幅改善组织。 对可用数据进行快速群集和划分的能力，有助于分析师和决策制定者接近实时地了解用户需要什么、部门运营的效力如何，或何时是群发电子邮件的最佳时机。 我们在两款解决方案上运行了基于 Hadoop 的机器学习工作负载和网络测试，发现 IBM Cloud 解决方案可以比 AWS 解决方案更快地群集数据、拥有更高吞吐量，并可以更快地传输数据。 如果您的组织需要获取大数据分析的优势，并在美国、英国和日本的办公室之间快速传输数据，IBM Cloud 解决方案可能是您组织的理想 IaaS 平台。

1 “Bluemix 现在变身为 IBM Cloud： 利用 170 多项服务，自信地进行构建”，2018 年 1 月 18 日，https://www.ibm.com/blogs/bluemix/2017/10/bluemix-is-now-ibm-cloud/

2017 年 8 月 4 日，我们最终确定了我们测试的硬件和软件配置。 最新和最近发布的软硬件常常出现更新，因此，不可避免的，当本报告发表时，这些配置可能无法代表最新版本。 对于较旧的系统，我们选择代表这些系统典型采购的配置。 我们在 2017 年 8 月 13 日完成了实际测试。

附录 A：系统配置信息

服务器配置信息	AWS 服务器 - Xen HVM domU
BIOS 名称和版本	Xen 4.2
操作系统名称和版本/内部版本号	Ubuntu 16.04.3 LTS
上次操作系统更新/打补丁的日期	07/20/2017
处理器	
处理器数量	1
供应商和型号	Intel Xeon 处理器 E5-2676 v3
内核数量（每个处理器）	40
内核频率 (GHz)	2.40
步进	2
内存模块	
系统总内存 (GB)	160
内存模块数量	10
大小 (GB)	16
存储控制器	
供应商和型号	Intel 82371SB PIIX3 IDE
本地存储	
驱动器 #1 大小 (GB)	100
驱动器 #2 大小 (GB)	500
网络适配器	
供应商和型号	Intel 82599 以太网控制器

服务器配置信息	IBM Cloud 服务器
操作系统名称和版本/内部版本号	Ubuntu 16.04.3 LTS
上次操作系统更新/打补丁的日期	07/20/2017
处理器	
处理器数量	1
供应商和型号	Intel Xeon 处理器 E5-2683 v4
内核数量（每个处理器）	32
内核频率 (GHz)	2.10
步进	1
内存模块	
系统总内存 (GB)	128
存储控制器	
供应商和型号	Intel 82371SB PIIX3 IDE
本地存储	
驱动器 #1 大小 (GB)	350
驱动器 #2 大小 (GB)	500

附录 B：我们如何测试

设计比较情形

对于本测试中的比较研究部分，我们通过测量直接、真实的 CPU 密集型工作负载性能，比较了 IBM Cloud 配置与 AWS 配置。 我们使用了 Intel HiBench Hadoop 基准测试套件中的 K-means Clustering 工具。 我们选择这款工具以凸显在平均计算吞吐量方面的差距，而非利用合成（因此客观上用处较小）基准测试工具来测量在有限任务中的 CPU 性能。

我们对云进行了配置以获得类似的性能，从而公平地评估每个解决方案的计算吞吐量 由于供应商配置解决方案的局限性，我们无法实现完全匹配的配置。 以下部分概括了关键的共性和差异。

通用配置

在每个云中，节点充当四个角色之一：客户端（基准测试机器）、配置（配置管理节点）、基础架构（Hadoop 名称节点，也称主节点）或从节点（Hadoop 数据节点，即被测试的系统） 我们创建拥有四个内核和 16 GiB 的虚拟机，以满足客户端、配置和基础架构角色的要求。对于每个虚拟机，我们分配了充足的存储，以支持基准测试和 Hadoop 基础架构。

在 ESXi 中创建首个虚拟机

所有从节点使用 Intel Xeon 处理器 E5-2600 v3（虚拟化）。

IBM Cloud

- 每个从节点有 32 个线程（一个 CPU x 32 个内核/CPU x 一个线程/内核）和 128 GiB RAM。
- 云解决方案在四个节点拥有 128 个线程，总计 512 GiB RAM。

AWS

- 每个从节点有 40 个线程（一个 CPU x 40 个内核/CPU x 1 个线程/内核）和 160 GiB RAM。
- 云解决方案在三个节点拥有 120 个线程，总计 480 GiB RAM。

联网

在供应商配置选项允许的范围内，所有节点都应尽可能近得共处一地，并配置尽可能快的网络。 对于 IBM Cloud 解决方案，所有节点都位于 Dallas 06 中，其中虚拟机采用 1Gbps 网络。 对于 AWS 解决方案，我们指定了一个通用放置组，让节点共处一地并为每个节点分配 10Gbps 网络。

存储

对于客户端、配置和基础架构三种节点，我们为节点配置了 25GB 一般性存储主驱动器。 IBM Cloud 解决方案使用基于存储区域网络 (SAN) 的驱动器。 我们用 EBS 配置了 AWS 解决方案（特别是一般用途的 SSD (GP2)）。

对于从节点，我们为每个节点提供一个更大的主/操作系统驱动器和一个大得多的驱动器，以支持 Hadoop HDFS。 虚拟 IBM Cloud 节点的驱动器是 SAN 驱动器（100 GB 用于操作系统，500 GB 用于 HDFS）。在 3,000 IOPS IO1 存储上，我们为 AWS 解决方案的从节点配置了 100 GB 用于操作系统，500 GB 用于 HDFS。

方法

按照以下通用过程测试两款解决方案：

1. 为虚拟机配置 Linux Ubuntu 16.04 操作系统。
2. 在基础架构节点和从节点上，使用 Ambari 安装并配置 Hadoop。
3. 在客户端节点上，安装并配置 Intel HiBench。
4. 在客户端节点上，安装并配置 IBM LPCPU 监测实用程序。
5. 准备 HiBench 工作负载。
6. 为测试调节从节点。
7. 在从节点上，启动 LPCPU 监测。
8. 运行 HiBench Benchmark 并收集结果。
9. 在从节点上，停止 LPCPU 监测并收集结果。
10. 对于每项复制，完成第5 步至第 9 步。

为虚拟机配置 Linux Ubuntu 16.04

每个供应商的配置机制略有不同，但遵从相同的通用模式：

1. 登录 Web 门户。
2. 单击设备或实例。
3. 为以下各项选择合适的值：
   - 数量
   - 区域组或放置组
   - 操作系统
   - CPU
   - 内存
   - 联网
   - 磁盘
4. 确认预留并等待虚拟机启动。
5. 使用 SSH 登录虚拟机，并配置任何软件必备条件，以支持 Hadoop、Ambari、HiBench、LPCPU 或任何配置管理软件。
6. 对于集群中的每个主机，执行以下步骤：
   a. 连接到主机 <XXX>：
      ```
      > ssh root@XXX
      ```
   b. 为根用户创建 RSA 密匙对：
      ```
      > ssh-keygen -t rsa -b 8192 -N "" -f ~/.ssh/id_rsa
      ```
   c. 对于集群中的其他每个主机 YYY，将 root@XXX 的 SSH 密钥复制到 YYY：
      ```
      > ssh-copy-id -i ~/.ssh/id_rsa.pub root@YYY
      ```
   d. 为集群中的每个主机重复第 6c 步。
7. 为集群中的每个主机重复第 6 步。

安装和配置 Hadoop

1. 在所有节点上，安装 Ambari REPO 和密钥：
   ```
   > 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. 更新程序包缓存，并应用升级：
   ```
   > apt-get update -y
   > apt-get upgrade -y
   ```
3. 在配置节点中，安装 Ambari Server：
   ```
   > apt-get install -y ambari-server
   > ambari-server setup -j PATH_TO_JAVA_HOME -s
   > ambari-server restart
   ```
4. 在其他节点中，安装 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. 在配置节点中，计算或设置以下加粗的变量，如下所示：
   - 集群中从节点的数量：
     NUMBER_OF_SLAVES = **X**
   - 任何节点上的最小内存容量 (MiB)：
     MIN_MEM = **131072**
   - 为系统预留的内存 MB（例如 28 GB）：
     OS_RESERVE_MB = **31072**
   - 每个节点的线程数量：
     NUM_CORES = **num_cpus * cores/cpu * threads/core**
   - 计算允许的内核数量：
     YARN_NUM_CORES = **NUM_CORES - 2**
   - 允许的 Yarn 内存：
     YARN_MEMORY_MB = **MIN_MEM - OS_RESERVE_MB = 100000**

- Yarn 内存分配器：
  YARN_MIN_ALLOC_MB = **8192**
  YARN_MAX_ALLOC_MB = **32768**
  YARN_INCREMENT_MB = **512**
- Yarn vcpu 分配器：
  YARN_MIN_CORES = **1**
  YARN_MAX_CORES = **30**
  YARN_INCREMENT_CORES = **1**
- MapReduce 内存：
  MAPRED_MAP_MEMORY_MB = **4096**
  MAPRED_REDUCE_MEMORY_MB = **8192**

a. 在 Ambari 中创建将主机映射到 Hadoop 角色的“hostmap”：

```
> 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. 创建 Ambari 蓝图：

```
> 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. 注册蓝图

```
> 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
```

注意：您还可以执行一个 GET CURL 请求到 http://${AMBARI_SERVER}:8080/api/v1/blueprints/${BLUEPRINT_NAME} 以确认您已成功注册蓝图。

d. 注册集群 (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. 使用浏览器，导航至 AMBARI_SERVER:8080，登录并监测运行状态，直到集群部署完成为止。

注意：您还可以执行一个 GET CURL 请求到 http://${AMBARI_SERVER}:8080/api/v1/clusters/${CLUSTER_NAME}/requests/1 以监测部署的状态。

安装 HiBench

在客户端节点上，执行以下步骤：

1. 安装必备条件：

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

2. 克隆 HiBench：

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

3. 构建 HiBench：

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

4. 停用 SSH StrictHostKeyChecking：

```
> nano ~/.ssh/config

 Host *
    StrictHostKeyChecking no

 [...]
```

注意： 第 4 步让您可以生成监测/利用情况报告。

安装 LPCPU

在从节点上，执行以下步骤：

1. 安装必备条件：

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

2. 下载 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. 提取 LPCPU：

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

准备 HiBench

1. 计算或记录以下加粗的数量：

- HDFS 用户名称
  HDFS_USER = **XXX**
- 第一个主节点完全限定的域名
  MASTER_1_FQDN = **master1.my_cloud.org**
- 所有主节点完全限定的域名，用空格分隔
  MASTER_FQDNS = **master1.my_cloud.org master2.my_cloud.org master3.my_cloud.org**
- 所有从节点完全限定的域名，用空格分隔
  SLAVE_FQDNS = **slave1.my_cloud.org slave2.my_cloud.org slave3.my_cloud.org**
- 工作负载的大小 tiny、huge、gigantic、bigdata 等
  HIBENCH_SCALE = **gigantic**
- 从节点数量
  S = **从节点的总数**
- 线程数量
  C = **所有从节点中 CPU/线程的数量**
- 任何从节点上的最小内存，单位：GiB
  U = **128**
- 要占用内存的分数，0.9（90%）是个好数值
  W: **0**
- 主节点数量
  M: **3**

- Spark 内核数量（5 是合理值）
  EXECUTOR_CORES = **5**
- Spark 执行器数量
  EXECUTOR_NUM = **floor((C-S) / EXECUTOR_CORES)**
- Spark 内存
  EXECUTOR_MEMORY = **floor((U*S) / EXECUTOR_NUM)**
- Spark 内存
  DRIVER_MEMORY = **floor((U*S) / EXECUTOR_NUM)**
- Spark map 并行处理
  MAP_PARALLELISM = **C-S**
- Spark reduce 并行处理
  SHUFFLE_PARALLELISM = **C-S**

2. 设置 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. 设置 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. 设置 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. 初始化存储：

   a. 如果存储此前已初始化，清除它：

```
> 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. 创建必需目录：

```
> 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. 运行准备脚本：

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

调整从节点

在所有从节点上，执行以下步骤：

1. 停用交换：

```
> swapoff -a
```

2. 重新启用交换：

```
> swapon -a
```

3. 刷新页面缓存：

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

注意： 您可以在所有节点上执行这项操作，但我们在测试中仅将目标锁定从节点。

启动 LPCPU 监测

在所有从节点上，执行以下步骤：

1. 启动 LPCPU 监测：

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

注意： 在测试期间让 SSH 会话开启、让 LPCPU 运行（或包装到 start-stop-daemon 等以在后台运行）。

运行 HiBench K-means benchmark

在客户端节点上，执行以下步骤：

1. 运行基准测试：

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

2. 收集 ~/HiBench/report 的内容。

停止 LPCPU 监测

在所有从节点上，执行以下步骤：

1. 要在 SSH 会话中打断 lpcpu.sh，输入 `[CTRL] + C`。
2. 等待 tarball 创建好 (output/data.tar.bz2)。
3. 提取 tarball 并运行 postprocess.sh 脚本。
4. 收集 output/data 目录的内容。

测量带宽

我们使用基于 TCP 的 ping 和安全复制工具，并选择了三个地区（美国东海岸、英国和日本），测量其中一个地区的虚拟机与其中另一个地区的虚拟机之间的网络性能，以了解网络延迟和吞吐量性能。 我们使用 paping 执行测试，测试了两项服务的上述方面。

通过连接到目标云环境上运行的 Web 服务器，并使用 ping 实用程序测量到 80 端口的 30 秒内的平均延迟，我们执行了 TCP-ping 测试，然后取三次测试的中值结果。

我们使用 scp 测试了文件复制性能，并统计了传输 25MB 文件和 500MB 文件以及包含 100MB 小文件的文件的持续时间。

云实例配置

每项服务三个虚拟机，美国东部、西欧和亚太（日本），配置如下：

区域

- AWS：北维吉尼亚州、爱尔兰、亚太（东京）
- IBM Cloud：华盛顿特区、伦敦、东京

虚拟机实例类型

- 所有虚拟机运行 Ubuntu 14。

虚拟机配置

- AWS EC2： “m4.xlarge”通用映像，包含 4 个 vCPU、16GB RAM、EBS 存储、“高”网络性能
- IBM Cloud：计算虚拟服务器，配置是 4 个 2.0GHz 内核、16GB RAM、1 Gbps 公共及私有网络上行链路

本项目受 IBM 委托。

阅读英文版原始报告，网址： http://facts.pt/MVyW7v.

Principled Technologies®

事实至关重要®