Open Cloud Artificial Intelligence (AI) Reference Design using Intel® Products

White Paper

November 2018
This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.

Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

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: http://www.intel.com/design/literature.htm

Arria, Intel, the Intel logo, Intel Saffron, Intel Xeon Phi, Movidius, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2018, Intel Corporation. All rights reserved.
# Contents

## 1.0 Introduction ............................................................................................................................6
  1.1 Platform Introduction.............................................................................................................6
  1.2 Infrastructure Deployment Architecture..............................................................................7
  1.3 Cloud Platform Services and Functions .............................................................................10

## 2.0 Reference Design Details .......................................................................................................12

## 3.0 Configuring Hardware for AI ..................................................................................................14
  3.1 Computation .........................................................................................................................14
  3.2 Network ...............................................................................................................................14
    3.2.1 Set up Intel® OP Software on CentOS* ........................................................................16
    3.2.2 Tuning IPoIB .................................................................................................................17
    3.2.3 Tuning TCP Parameters ..............................................................................................17
    3.2.4 Testing IPoIB over Intel® OPA ....................................................................................18
  3.3 Storage ..................................................................................................................................19

## 4.0 AI Computing Suites .............................................................................................................20
  4.1 Single-node Intel® Optimization for Caffe* ........................................................................20
  4.2 Cluster-based Intel® Optimization for Caffe* .....................................................................21
  4.3 TensorFlow* .........................................................................................................................22

## 5.0 System Architecture of Cloud Service .....................................................................................24
  5.1 Cloud Service System ...........................................................................................................24
    5.1.1 Cloud Service System Overview ................................................................................24
    5.1.2 Setting up Cloud Service System ................................................................................25
  5.2 Network System ...................................................................................................................29
    5.2.1 Network Overview .........................................................................................................29
    5.2.2 Deploying Calico Network in Kubernetes* ..................................................................30
  5.3 Storage Systems ...................................................................................................................31
    5.3.1 Ceph Architecture Overview .......................................................................................31
    5.3.2 Ceph Deployment ..........................................................................................................33
      5.3.2.1 Setting up Ceph .......................................................................................................34
      5.3.2.2 Installing Ceph ........................................................................................................35
    5.3.3 GlusterFS* Architecture Overview ..............................................................................36
    5.3.4 GlusterFS* Deployment ...............................................................................................37
    5.3.5 GlusterFS* on Kubernetes* ........................................................................................37
  5.4 AI Frameworks ......................................................................................................................39
    5.4.1 Running Intel® Optimization for Caffe* on Kubernetes* ..............................................39
    5.4.2 TensorFlow* on Kubernetes* .......................................................................................40

## 6.0 Intel AI Edge Devices and Applications ....................................................................................42
  6.1 Intel® Movidius™ NCS Overview .........................................................................................42
    6.1.1 Installation ......................................................................................................................43
6.1.2 Intel® Movidius™ Neural Compute SDK Tools ......................................................... 44
6.1.3 Neural Compute API (NCAPI) .................................................................................. 45
6.1.4 Intel® Movidius™ NCSDK Examples ........................................................................ 45
6.1.5 Use Case: Object Detection and Classification with Intel® Movidius™ NCS .......... 46

Appendix A References .................................................................................................. 47

Figures

Figure 1. China AI Open Innovation Platform Functional Architecture .............................. 7
Figure 2. Basic Architecture of AI Open Innovation Platform ........................................... 9
Figure 3. Overall Architecture of China AI Open Platform .............................................. 10
Figure 4. Network Deployment of Reference Verification Platform .................................. 12
Figure 5. Intel® OPA Component Architecture .................................................................. 15
Figure 6. MPI One-to-Many Communication Mode ....................................................... 21
Figure 7. MPI Many-to-One Communication Mode ......................................................... 22
Figure 8. MPI Many-to-Many Communication Mode ...................................................... 22
Figure 9. Overall Cloud Service Architecture .................................................................. 25
Figure 10. Network Deployment Architecture in Kubernetes* ......................................... 30
Figure 11. Ceph Component Architecture ........................................................................ 31
Figure 12. Ceph Storage Architecture ............................................................................. 32
Figure 13. SSD Caching Architecture at Ceph Client ....................................................... 33
Figure 14. Replicated Volume Mode of GlusterFS* .......................................................... 36
Figure 15. Application of GlusterFS* on Kubernetes* ..................................................... 39
Figure 16. Intel® Optimization for Caffe* on Kubernetes* ............................................... 40
Figure 17. TensorFlow* on Kubernetes* ......................................................................... 41
Figure 18. Myriad 2 Chip Structure .................................................................................. 42
Figure 19. Intel® Movidius™ Development Process ....................................................... 43
Figure 20. Object Detection and Classification with Intel® Movidius™ NCS ...................... 46

Tables

Table 1. Hardware Deployment for China AI Open Innovation Platform ............................... 8
Table 2. Software Deployment of Reference Verification Platform ..................................... 13
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2018</td>
<td>001</td>
<td>No technical changes. Clerical correction of references to Figure 7 and Figure 8.</td>
</tr>
<tr>
<td>October 2018</td>
<td>001</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

§
1.0 Introduction

This white paper describes the architecture, design details, and configuration for an AI reference design based on Intel® products. The reference design was developed for the China National AI Innovation Competition.

1.1 Platform Introduction

The China AI Open Innovation Platform was built on Intel AI hardware platforms and comprehensive solutions, including the following aspects:

AI Hardware Infrastructure

We constructed an AI hardware infrastructure consisting of high performance servers with Intel® Xeon® Platinum processors and Intel® Xeon Phi™ technology, Intel® Omni-Path Architecture (Intel® OPA) 100Gb high speed network interconnection devices, Intel SSD high speed flash storage devices, and security devices, to meet the needs of massive data access, storage, and intelligent processing, and to satisfy requirements for safe and reliable operations.

AI Cloud Platform

By adopting open source cloud platform architecture, including KVM*, Docker*, OpenStack*, Kubernetes*, CEPH*, and GlusterFS* distributed storage architecture, we performed virtualization for hardware facilities to form a virtual-level resource pool system. Each application system was provided with basic IT resources on demand, including computing capability, storage capacity and networking capability to quickly adapt to dynamically changing business requirements and achieve resilient resource allocation capabilities.

Using AI development frameworks optimized for Intel platforms, including Intel® Optimization for Caffe*, TensorFlow*, and others, we achieved high performance, high scalable and distributed AI training frameworks. The training frameworks used AI libraries and tools optimized for Intel platforms, including Intel® MKL, Intel® MKL-DNN, and BigDL, to maximize the performance of hardware platforms.

A unified Web interface provided central management for all the hardware resources, including virtual machines, resource pools, networks, storages, containers, bare metals, AI development frameworks, and others, which then allocated various types of software and hardware resources for AI development.
**AI Development Suite Environment**

We integrated AI suites, including Intel FPGA development boards, Intel® Movidius™ compute sticks, desktop computers, peripherals, and others, to build an AI development suite environment. The environment enabled the development teams to develop AI inference sides, mobile and low-power scenarios, and display their results. The environment also provided front-end equipment support for the residing platform teams.

**Figure 1. China AI Open Innovation Platform Functional Architecture**

![China AI Open Innovation Platform Functional Architecture](image)

---

**1.2 Infrastructure Deployment Architecture**

The infrastructure hardware included Intel® Xeon® and Intel® Xeon Phi™ platform servers for AI workspaces, Intel® Xeon® E5 platform servers for storage and node management, Intel® Omni-Path 100G for high speed interconnection in AI workspaces, and Intel® Arria® 10 GX FPGA and Intel® Movidius™ Neural Compute Stick (NCS) for AI deployment ends. The following table shows the actual configuration.
Table 1. **Hardware Deployment for China AI Open Innovation Platform**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Servers</strong></td>
<td></td>
</tr>
<tr>
<td>High Performance Intel® Xeon® Servers</td>
<td>CPU: Intel® Xeon® Platinum 8180*2</td>
</tr>
<tr>
<td></td>
<td>MEM: 64GB* DDR4 2400 * 24</td>
</tr>
<tr>
<td></td>
<td>DISK: SATA SSD Intel S3610 1.6TB*2</td>
</tr>
<tr>
<td></td>
<td>Network port: Gigabit<em>4 ports, 10 Gigabit</em>4 ports, Intel® OPA 100Gb*1</td>
</tr>
<tr>
<td></td>
<td>IPMI: supported</td>
</tr>
<tr>
<td>High Performance Intel® Xeon Phi™ Servers</td>
<td>CPU: Intel® Xeon Phi™ 7250, 16GB MCDRAM included *4 (codename KNL)</td>
</tr>
<tr>
<td></td>
<td>MEM: 64GB DDR4 2400 *24</td>
</tr>
<tr>
<td></td>
<td>DISK: SSD 1.6TB*12 SSD Micron 5100 1.92T</td>
</tr>
<tr>
<td></td>
<td>Network port: Gigabit<em>8 ports, 10 Gigabit</em>8 ports, Intel® OPA 100Gb*4</td>
</tr>
<tr>
<td></td>
<td>IPMI: supported</td>
</tr>
<tr>
<td>Platform Management Servers</td>
<td>CPU: Intel® Xeon® E5-2650V3 *2</td>
</tr>
<tr>
<td></td>
<td>MEM: 32GB DDR4 2400 *8</td>
</tr>
<tr>
<td></td>
<td>DISK: SSD 800GB<em>2, 900GB 15000rpm SAS</em>6</td>
</tr>
<tr>
<td></td>
<td>Network port: Gigabit<em>4 ports, 10 Gigabit</em>4 ports</td>
</tr>
<tr>
<td></td>
<td>RAID card: not less than 1G cache</td>
</tr>
<tr>
<td></td>
<td>IPMI: supported</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td></td>
</tr>
<tr>
<td>Intel® OPA 100Gb Switch</td>
<td>48-port 100Gb Intel® OPA switch</td>
</tr>
<tr>
<td><strong>AI Suite</strong></td>
<td></td>
</tr>
<tr>
<td>Intel® Movidius™ Compute Stick</td>
<td>Intel® Movidius™ Neural Compute Stick (NCS)</td>
</tr>
</tbody>
</table>

The deployment architecture is shown in the following figure.
The basic architecture of the reference design was divided into three areas:

**Internet Boundary Security Zone**

The primary functions of this area were providing external ports for services, ensuring access quality through dual links of China Telecom and China Unicom, and ensuring the safety of the platform with a series of front-end security equipment.

**AI Cloud Production Area**

The AI Cloud Production Area was divided into compute resource pools and storage resource pools. Compute resource pools provided high performance computing power with Intel® Xeon® Platinum processors and Intel® Xeon Phi™ technology. Storage resource pools provided effective distributed capacity. Both compute resource pools and storage resource pools used Intel® OPA 100Gb to provide maximum bandwidth and lowest latency for inference learning and model training. The access layer and convergence layer used 10Gb and 40Gb respectively for high speed interconnection.
Security Management Area

The Security Management Area performed unified division management for a series of security equipment, including cloud management platform, anti-virus system, security management servers, vulnerability scanning servers, and others, to ensure the security of production areas.

1.3 Cloud Platform Services and Functions

Conceptually, the platform was divided into layers that map to business requirements: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Each layer provided safe and reliable AI hardware resources and development kit services for China's AI research and development teams, related enterprises, and social organizations. The overall architecture is shown in the following figure.

Figure 3. Overall Architecture of China AI Open Platform
**IaaS layer**: This layer conducted unified hardware management, by creating high performance server clusters and large scale distributed storage clusters, using 100GB network high speed interconnection, and adopting OpenStack, Kubernetes, and Ceph. It provided IaaS services externally, including virtual machines, bare-metals, storage, and network functions.

**PaaS layer**: Unlike traditional PaaS scenarios, AI development platforms require the PaaS layer to be able to serve the AI developers themselves. The reference design used IaaS features to create services such as containers, databases, cache and big data service required by the PaaS layer. The development test platform (DevOps in the diagram), was created to meet the requirements of fast application deployment, testing and packaging, and application management, enabling developers to focus on applications.

**SaaS layer**: This layer provided the developer portal and administrator portal, facilitating rapid application and use of resources by AI development teams, and unified operation and maintenance management of the platform by administrators.
2.0 Reference Design Details

This section describes the hardware, network diagram, and software configuration for the reference design.

For hardware details, refer to Table 1.

The following figure shows the deployment network architecture of this platform.

Figure 4. Network Deployment of Reference Verification Platform

The software configuration and system plan are shown in the following table.
<table>
<thead>
<tr>
<th>Table 2. Software Deployment of Reference Verification Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Version</strong></td>
</tr>
<tr>
<td><strong>Kernel Version</strong></td>
</tr>
<tr>
<td><em><em>OpenStack</em> Version</em>*</td>
</tr>
<tr>
<td><em><em>Kubernetes</em> Version</em>*</td>
</tr>
<tr>
<td><em><em>Ceph</em> Version</em>*</td>
</tr>
<tr>
<td><strong>Disk Partitioning</strong></td>
</tr>
<tr>
<td><strong>System Disk</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>System Software Package Group</strong></td>
</tr>
</tbody>
</table>
3.0 Configuring Hardware for AI

This section describes the requirements of AI applications for various IT resources and the construction and optimization of corresponding hardware platforms.

3.1 Computation

The core of computation in the field of AI is large-scale matrix operations, which requires that computing units are able to perform parallel and scalable processing, as well as optimization for software layers.

Intel® Xeon® Scalable processors are a broadly-used platform for AI that can provide high throughput for inference and training. Compared with Intel® Xeon® E5 systems, the Intel® Xeon® Scalable processor platform can increase inference throughput up to 22 times and training throughput up to 18 times. See https://www.intel.cn/content/www/cn/zh/benchmarks/server/xeon-scalable/xeon-scalable-artificial-intelligence.html

The Intel® Xeon® Scalable processor family contains products at different levels, from the highest-level product (Intel® Xeon® Platinum 8100 processor) to the entry-level product (Intel® Xeon® Bronze 3100 processor). The Intel® Xeon® Scalable processor family provides a variety of performance, scalability, and feature options, and supports versatile workloads of data centers. See https://www.intel.cn/content/dam/www/public/cn/zh/documents/product-briefs/xeon-scalable-platform-brief-cn.pdf

3.2 Network

Training and inference are two separate steps in machine learning. Traditional training takes days or even weeks. However, inference is almost real-time. Improving the accuracy of inference requires repeated and fast training. Therefore, training must be completed in a shorter time period, such as a few hours. This requires an innovative platform to provide parallel and highly scalable computing.

Important network factors that were considered when designing the reference platform include:

- Scalability: ideally, the throughput and performance of training increases linearly with the number of computing nodes.
- High bandwidth and low latency communication network between computing nodes.
- High bandwidth and low latency storage network.
We selected Intel® Omni-Path Architecture (Intel® OPA) for the reference design because it meets the network criteria. According to a research report, using Intel® OPA in the field of machine learning can achieve more than 90% linear scalability. See https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/omni-path-white-paper-final-december.pdf

Intel® OPA is one component of the Intel® Scalable System Framework. It provides high performance, scalable, and low cost fabric communication for HPC. This end-to-end fabric solution provides 100 Gbps of port bandwidth while providing low latency, optimized packet protocol, dynamic traffic management, and advanced Quality of Service (QoS) even in super-large scales, as shown in the following figure.

**Figure 5. Intel® OPA Component Architecture**

In this architecture:

- **HFI (Host Fabric Interface):** Provides fabric connectivity for compute, service and management nodes.
- **Switches:** Permits creation of various network topologies to connect a scalable number of endpoints.
- **Fabric Manager:** Provides centralized provisioning and monitoring of fabric resources.

The following sections describe how the network was set up for the reference design. It includes commands for configuring Intel® Omni-Path Software (Intel® OP Software) for proper operation.
3.2.1 Set up Intel® OP Software on CentOS*


1. Confirm the Intel® Omni-Path Host Fabric Interfaces (HFIs) in the system:
   # yum install -y pciutils
   # lspci -vv | grep Omni

   18:00.0 Fabric controller: Intel Corporation Omni-Path HFI
   Silicon 100 Series [discrete] (rev 11)
   Subsystem: Intel Corporation Omni-Path HFI Silicon 100 Series
   [discrete]

2. Download Intel® Omni-Path Software:

   Intel® Omni-Path Software can be installed by downloading the installation package or from package source. The installation package was downloaded from the following link:

   https://downloadcenter.intel.com/search?keyword=Omni-Path

   Since Intel® Omni-Path Software is packaged into package source of CentOS, we installed the software from package source.

3. Install prerequisites:

   # yum install -y libibmad libibverbs librdmacm libibcm qperf
   perfptest rdma infinipath-psm expat elfutils-libelf-devel
   libstdcxx++-devel gcc-gfortran atlas tcl expect tcsh sysfsutils
   pciutils bc libibumad libibumad-devel libibumad-devel libibumad-
   devel libibverbs-devel libibmad-devel librdmacm-devel ibacm-
   devel openssl-devel libuuid-devel expat-devel infinipath-psm-
  -devel valgrind-devel libgnome libibverbs* opensm-libs libhfill
   papi ncurses-devel hwloc hwloc-gui

4. Install Intel® Omni-Path Software and reboot computer:

   # yum install -y opa-basic-tools
   # reboot

5. Confirm that the HFI driver hfi1 has been installed and loaded:

   # modprobe hfi1
   $ lsmod | grep hfi1

6. Configure IPoIB IPV4:

   # vi /etc/sysconfig/network-scripts/ifcfg-ib0
   DEVICE=ib0
   BOOTPROTO=static
   IPPADDR=192.168.200.119
   BROADCAST=192.168.200.255
   NETWORK=192.168.200.0
   NETMASK=255.255.255.0
   ONBOOT=yes
   CONNECTED_MODE=yes
   MTU=65520
7. Install Intel® OPA Fabric Manager (FM):
   
   ```bash
   yum install -y opa-fastfabric opa-address-resolution opa-fm
   # systemctl start opafm
   ```

8. Start IPoIB interface:
   
   ```bash
   # ifup ib0
   ```

To verify that RDMA is working normally, refer to
http://www.rdmamojo.com/2015/01/24/verify-rdma-working/

### 3.2.2 Tuning IPoIB

Intel® Omni-Path products send Internet Protocol (IP) traffic over the fabric (IPoFabric). This mechanism is also referred to as IP over IB or IPoIB. From a software point of view, IPoFabric and IPoIB behave the same way and, in fact, use the same ib_ipoib driver to send IP traffic over the ib0 and/or ib1 ports.

This section describes how we configured IPoIB for the reference design.

Enable Transparent Huge Pages:

```bash
$ cat /sys/kernel/mm/transparent_hugepage/enabled [always]
madvise never
```

Do Not Enable intel_iommu:

```bash
# cat /etc/default/grub
```

Delete iommu setting from this line:

```
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=centos/root rd.lvm.lv=centos/swap rhgb quiet"
```

Display driver parameters:

```bash
for x in /sys/module/hfi1/parameters/*; do echo "$(basename $x) " $(cat $x); done
```

Set 8K MTU:

```bash
cat /etc/sysconfig/network-scripts/ifcfg-ib0
ONBOOT=yes
NM_CONTROLLED=no
MTU=65520
CONNECTED_MODE=yes
```

Set "IPoFabric Datagram Mode" to "connected" for the best performance.

```
[root@jfz1r04h19 z1r04h19]# cat /sys/class/net/ib0/mode
connected
```

### 3.2.3 Tuning TCP Parameters

Generally, it is not necessary to tune TCP parameters. If memory is abundant, setting TCP parameters typically improves performance by 10%.

To tune TCP parameters, enter the following commands:
3.2.4 Testing IPoIB over Intel® OPA

This section describes how we tested IPoIB. We used iperf3 as the bandwidth testing tool and ib_send_lat as the latency testing tool. The bandwidth for TCP mode is 25G.

```
[z1r04h19@jfz1r04h19 ~]$ iperf3 -c 192.168.200.118
Connecting to host 192.168.200.118, port 5201
```

### Bandwidth for TCP Mode

<table>
<thead>
<tr>
<th>Interval</th>
<th>Transfer</th>
<th>Bandwidth</th>
<th>Retr</th>
<th>Cwnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00-10.00 sec</td>
<td>29.3 GBytes</td>
<td>25.1 Gbits/sec</td>
<td>0</td>
<td>sender</td>
</tr>
<tr>
<td>0.00-10.00 sec</td>
<td>29.3 GBytes</td>
<td>25.1 Gbits/sec</td>
<td></td>
<td>receiver</td>
</tr>
</tbody>
</table>

The bandwidth for UDP mode is the same as that for TCP mode: 25G.

```
[z1r04h19@jfz1r04h19 ~]$ iperf3 -u -c 192.168.200.118 -b 40G
```

The default length of buffer is 128K. Setting it to 1M increased the bandwidth to 27.5G.

```
[z1r04h19@jfz1r04h19 ~]$ iperf3 -c 192.168.200.118 -l 1M
```

### CPU Affinity can reach 46G.

```
[root@jfz1r04h19 z1r04h19]$ iperf3 -c 192.168.200.118 -A 4
```

```
[z 4] 0.00-10.00 sec 54.2 GBytes 46.6 Gbits/sec 0 sender
```

### Latency Testing:

```
[root@jfz1r04h19 z1r04h19]$ ib_send_lat -a -c UD -d hfi1_0 -i 1 192.168.200.118
Max msg size in UD is MTU 4096
Changing to this MTU
```

```
-----------------------------------------------
Send Latency Test
Dual-port : OFF  Device : hfi1_0
Number of qps : 1  Transport type : IB
Connection type : UD  Using SRQ : OFF
TX depth : 1
Mtu : 4096[B]
Link type : IB
Max inline data : 0[B]
rdma_cm QPs : OFF
Data ex. method : Ethernet
-----------------------------------------------
local address: LID 0x01 QPN 0x0078 FSN 0xa16791
remote address: LID 0x05 QPN 0x00ea FSN 0xf8cb30
```
3.3 Storage

AI applications are always accompanied by data operations: from data collection to data flushing and preparation, to data being delivered to AI algorithm training and algorithm deployment, and data archiving and backup. Data has different requirements and focuses on storage and I/O at each phase. Refer to: https://blog.netapp.com/accelerate-i-o-for-your-deep-learning-pipeline/

This section focuses on the AI algorithm training and algorithm deployment phases.

Typically, AI algorithm training runs on a distributed, scalable system. Multiple computing nodes must obtain a large amount of data concurrently to provide to the computing units. Ideally, the speed of data supply should maintain 100% full-load computing for computing units (such as CPUs or GPUs) to speed up the algorithm training and reduce training time.

To achieve the above goal, storage has the following requirements in the training phase:

1. The storage system can provide highly scalable parallel I/O reading.
2. Storage media can provide IOPS with high bandwidth and low latency.
3. It can provide cache on the client side: AI algorithm training usually involves thousands or more iterations. Reading a large amount of data from disks for each iteration is difficult to meet the data needs for CPU processing. At the same time, it also requires the cache capacity as large as possible, and as much data as possible to be cached to local memory.

Modern distributed storage systems Ceph and GlusterFS both provide highly scalable parallel I/O reading. Refer to Section 5.3, Storage Systems for more details.

Providing high bandwidth and low latency IOPS requires support from underlying storage media and network connections. Compared with traditional HDDs, modern SSDs offer higher IOPS, lower read-write latency, and higher reliability,
4.0 AI Computing Suites

This section describes several AI computing suites that are available on the reference design.

4.1 Single-node Intel® Optimization for Caffe*

Caffe* is an open source deep learning framework that was originally developed by Berkeley Vision and Learning Center, BVLC. Intel® Optimization for Caffe* is specifically modified for Intel architecture. The solution (https://github.com/intel/caffe) supports Intel® Xeon® processors and Intel® Xeon Phi™ processors, among others. This version of Caffe integrates the latest release of Intel® Math Kernel Library, is optimized for Intel® Advanced Vector Extensions 2 and Intel Advanced® Vector Extensions 512 instructions, and enables parallel optimization computing for multi-core based on OpenMP*. In addition, Intel® Optimization for Caffe* supports multi-node distributed computing and supports Intel® Omni-Path Architecture, enabling highly scalable multi-node computing. For details, see https://software.intel.com/sites/default/files/managed/bd/7d/Caffe_optimized_for_IA.pdf

To use Intel® Optimization for Caffe*, perform the following steps.

1. Set up the Intel® MKL as the BLAS library in the compilation configuration file, Makefile.config.
   # BLAS choice:
   # mkl for MKL
   # open for OpenBlas
   BLAS := mkl

2. Configure the locations of include and lib of the Intel® MKL.

3. Add 'engine:"MKL2017"' to the file train_val.prototxt or solver.prototxt. For example:
   lenet_train_test_mlsl.prototxt: engine: MKL2017;

Alternatively, add option caffe tool: -engine "MKL2017" during run-time.

For detailed steps, see https://github.com/intel/caffe/wiki/Multinode-guide

In the reference design, we used the method of Intel® Optimization for Caffe* Docker image as follows:

$ docker pull bvlc/caffe:intel
$ docker run -it bvlc/caffe:intel /bin/bash

If you want to build the Intel® Optimization for Caffe* Docker image by yourself, see https://github.com/intel/caffe/wiki/How-to-build-run-Intel-Caffe-Docker-image
4.2 Cluster-based Intel® Optimization for Caffe*

Native Caffe does not support distributed computing and can only use one CPU core from multiple CPU cores for single computing node. Intel® Optimization for Caffe* uses the Intel® Machine Learning Scaling Library (Intel® MLSL) to provide distributed training. In this scenario, Intel® MLSL is built on top of MPI, and includes the Intel® MLSL Software Development Kit (SDK) and Intel® MPI Library Runtime component. The Intel® MLSL APIs support several deep learning frameworks, such as, Caffe*, Theano*, Torch*, and others. See https://github.com/intel/MLSL for details.

For machine learning tasks, MPI -- in addition to providing distributed training -- has the following advantages:

First, MPI has high performance aggregate communication implementations of Scatter, Gather, and Allreduce. In these implementations, MPI_Scatter and MPI_Bcast both are one-to-many communication mode, as shown in Figure 6. Their difference is that for the MPI_Bcast, process 0 sends the same piece of data to all processes, while MPI_Scatter sends different chunks of an array to all processes.

MPI_Gather takes elements from each process and gathers them into the root process, as shown in Figure 7. MPI_Allgather gathers all of the elements to each processes, as shown in Figure 8.

These MPI communication modes efficiently meet the communication requirements between distributed computing nodes in machine learning, especially in deep learning.

---

**Figure 6. MPI One-to-Many Communication Mode**

[Diagram showing MPI_Bcast and MPI_Scatter]
Secondly, MPI communication is implemented in a high performance network environment, such as Intel® OPA or InfiniBand®, which have highly optimized, leveraged RDMA network characteristics, and are capable of providing distributed communications with high bandwidth and low latency.

Thirdly, existing MPI programs can be easily ported to the Intel® Optimization for Caffe® computing framework.

MLSL provides data parallelism and model parallelism, supports SGD communication mode, and distributed weighted updates.

- Data Parallelism: partitions data sets, that is each computing node using the same computing model, but having different data batches.

- Model Parallelism: partitions tasks on computing model, that is each computing node having the same data set, but using different computing models.

Intel® Optimization for Caffe® adopts data parallelism. For more information on how to run multi-node Intel® Caffe®, see https://github.com/intel/caffe/wiki/Multinode-guide.

4.3 TensorFlow®

TensorFlow® (https://www.tensorflow.org/) is an open source computing framework developed by Google® that was first released in 2015. In 2017, Google officially
released version 1.0 of Google TensorFlow, and guaranteed that the API interfaces fully met the stability requirements for production environments. This is an important milestone for TensorFlow, indicating that it can be used safely in production environments. The TensorFlow computing framework supports various deep learning algorithms very well; however, its applications are not limited to deep learning.

As a leading deep learning and machine learning framework, TensorFlow* is critical for Intel and Google to unleash the best performance of Intel's hardware products. Engineers from Intel and Google worked closely and made significant optimizations for TensorFlow running on Intel CPUs. One of the most significant achievements was the adoption of Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN) in TensorFlow. Intel® MKL-DNN includes primitive acceleration required for the most popular networks for deep learning (including AlexNet, VGG, GoogLeNet and ResNet). These primitives include convolution, normalization, activation and inner product functions, and functions required for handling tensors.

The main points of the primitive acceleration include:

- Efficient execution of computations on Intel® architectures requires vectorization into the latest SIMD instructions (AVX2 or AVX512 for Intel® Xeon® processors).
- Efficient use of all cores, parallelism implemented in specific layers or operations, and cross-layer parallelism.
- Providing as much data as possible according to the needs of the execution units, balanced uses of prefetching and cache restriction techniques, and improving data formats for spatial and temporal localities. Multiple operations are combined to ensure efficient reuse of cache on CPUs.

Other optimizations include:

- Eliminate unnecessary and costly data layout conversions.
- Fuse multiple operations together to enable efficient cache reuse on CPUs.
- Handle intermediate states that allow for faster backpropagation.

For details, see https://software.intel.com/zh-cn/articles/tensorflow-optimizations-on-modern-intel-architecture
5.0 System Architecture of Cloud Service

Users can create distributed computing clusters of varying sizes according to the complexity of their own program computation. Generally speaking, users must determine the required number of parameter servers and computing servers to be created and the configurations of those servers. Typical configuration items include the number of CPUs, the number of GPUs, and memory capacity. After users have determined these configurations, the AI platform can quickly create distributed computing clusters that meet users' needs. Once the cluster creation is completed, the relevant information about the cluster creation is displayed through HTML pages. Users can write their own code to specify the requirement information and to assign the related parameters and computing tasks to the clusters.

This section describes the following systems:

- Cloud Service System
- Network System
- Storage Systems (Ceph and GlusterFS)
- AI Frameworks (Intel® Optimization for Caffe* and TensorFlow*)

5.1 Cloud Service System

5.1.1 Cloud Service System Overview

Although popular AI computing frameworks, such as TensorFlow, can sufficiently support AI training and inference, and can also support distributed computing, frameworks are somewhat inadequate in providing AI computing power in the form of cloud services and supporting large-scale computing deployments.

- TensorFlow's separate task resources cannot be isolated during training and it can lead to tasks interacting with each other due to resource preemption.
- TensorFlow lacks scheduling capability and requires users to manually configure and manage computing resources for tasks.
- For large clusters, the management of training tasks is troublesome. Significant development work must be done at the upper level to track and manage the status of each task.
- When users want to view the training logs of each task, they must find and SSH to the corresponding servers, which is inconvenient.

Using Kubernetes to load and schedule containers can provide a variety of resource management mechanisms such as ResourceQuota and LimitRanger, and can isolate resources between tasks. Its advantages include:
- Supporting configuration and scheduling of computing resources for tasks.
- Managing task status is simplified because training tasks are run as containers and Kubernetes provides a complete set of container PLEG interfaces.
- Viewing task logs using EFK/ELK and other logging schemes.
- Supporting distributed storage options with better reading performance (Ceph, GlusterFS)
- Enabling quick creation of a large-scale TensorFlow cluster through declarative files.

At the same time, as the foundation software for data center management, OpenStack can be used to manage various computing resources, from physical machines to virtual machines to containers. In this reference architecture, we used OpenStack's bare-metal management capability to:
- Transform the Kubernetes-based AI cloud that was enabled and optimized on the Intel platforms into a turn-key solution.
- Facilitate the installation in different environments.
- Decrease the requirements for deployment and use.

OpenStack can be extended to cloud environments that provide physical machines, virtual machines, and containers.

**Figure 9. Overall Cloud Service Architecture**

5.1.2 Setting up Cloud Service System

This section describes the following tasks that we completed for the reference design:
- Set up OpenStack Ironic* for bare-metal deployment.
- Create image for Kubernetes deployment.
Deploy the Intel® Optimization for Caffe® container image.

1. Deploying OpenStack Ironic node.

Since the nodes' primary purpose is to provision other physical machines, it is not necessary to deploy computing nodes for the virtual machines. The v-provision network (10.4.xx/16) was used as a provisioning network, which was used for PXE start of other nodes.

```bash
CONTROL_NODES="172.16.3.16"
API_IP_ADDRESS=$CONTROL_NODES
NET_NODES=$CONTROL_NODES
### all other networks except neutron physical mapping
API_NET="enp134s0f0"
## going to be the network controller&compute physical mapping interface
EXT_NET="enp61s0f0"
FEATURES="ironic"
```


Use a disk image builder to make an image containing packages such as Kubernetes and Intel® OPA. The image was used to deploy other physical machines.

```bash
# disk-image-create centos7 vm devuser selinux-disable epel-release clean-interface openssh-login opa-package k8s-package

Upload finished image to glance.

```bash
# glance image-create --name aicloud-image --visibility public

```

Upload deploy-kernel and deploy-ramdisk.

```bash
# glance image-create --name deploy-kernel --visibility public

```

```bash
# glance image-create --name deploy-ramdisk --visibility public

```

3. Registering physical nodes.

Nodes need to be registered first, according to IPMI information and actual hardware configuration.

```bash
# openstack baremetal node create --driver ipmi

```

```bash
--driver-info ipmi_username=$USER
--driver-info ipmi_password=$PASS
--driver-info ipmi_address=$ADDRESS
--driver-info deploy_kernel=${DEPLOY_KERNEL}
--driver-info deploy_ramdisk=${DEPLOY_RAMDISK}
--driver-info cleaning_network=$FLAT_NET
--driver-info provisioning_network=$FLAT_NET
```
Create ironic port.
# openstack port create $MAC_ADDRESS --node $NODE_UUID

Create resource class.
# openstack os-baremetal-api-version 1.21 baremetal node set $NODE_UUID
--resource-class BAREMETAL_AI

Set Ironic nodes to "available" state.
# openstack baremetal node validate $NODE_UUID
# openstack os-baremetal-api-version 1.11 node manage $NODE_UUID
# openstack Baremetal --os-baremetal-api-version 1.11 node provide $NODE_UUID

Create flavor according to the resource class set previously.
nova flavor-create ai-bm 10 $RAM_MB $DISK_GB $CPU
 nova flavor-key ai-bm set cpu_arch=$ARCH
 nova flavor-key ai-bm set resources:CUSTOM_BAREMETAL_AI=1
 nova flavor-key ai-bm set resources:Vcpu=0
 nova flavor-key ai-bm set resources:MEMORY_MB=0
 nova flavor-key ai-bm set resources:DISK_GB=0


Deploy physical machines with Ironic, and import script with user-data method to start Kubernetes master.
openstack server create --image centos-ai --flavor ai-bm --key-name ai-key --nic net-id=public1 -user-data k8s-master.sh ${NODENAME}

Start Kubernetes Minion.
openstack server create --image centos-ai --flavor ai-bm --key-name ai-key --nic net-id=public1 -user-data k8s-minion.sh ${NODENAME}

Based on the previously constructed Kubernetes environment, we deployed various AI frameworks to run tasks. Our goal was to easily deploy Intel® Optimization for Caffe* in a Kubernetes environment. To achieve this goal, we had to solve a few problems. In the process of making the image, we injected pre-set private/public key, avoiding more manual interactions for AI tasks running in isolated environments.

# cat Dockerfile
FROM ubuntu:16.04
RUN apt-get update & & apt-get install -y --no-install-recommends \
  cpio \
  build-essential \
  ...
```
cmake \
  git \
  wget \
  ssh \
  openssh-server \
  numactl \
  vim \
  net-tools \
  iputils-ping \
  screen \
  libmlx4-1 libmlx5-1 ibutils rdmacm-utils libibverbs1 \
  ibverbs-utils perf test infiniband-diags \
  openmpi-bin libopenmpi-dev \
  ufw \
  iptables \
  libboost-all-dev \
  libgflags-dev \
  libgoogle-glog-dev \
  libhdf5-serial-dev \
  libleveldb-dev \
  liblmdb-dev \
  libopencv-dev \
  libprotobuf-dev \
  libsnappy-dev \
  protobuf-compiler \
  python-dev \
  python-numpy \
  python-pip \
  python-setuptools \
  python-scipy & & \
  rm -rf /var/lib/apt/lists/*

ENV CAFFE_ROOT=/opt/caffe
WORKDIR $CAFFE_ROOT
# FIXME: clone a specific git tag and use ARG instead of ENV once
# DockerHub supports this.
ENV CLONE_TAG=1.1.1a
RUN pip install --upgrade pip
RUN git clone -b ${CLONE_TAG} --depth 1 
  https://github.com/intel/caffe.git . & & \
  for req in $(cat python/requirements.txt) pydot; do pip 
  install $req; done & & \
  mkdir build & & cd build & & \
  cmake -DCPU_ONLY=1 -DUSE_MLSL=1 -DCMAKE_BUILD_TYPE=Release .. 
  & & \
  make all -j"$(nproc)"

ENV MLSL_ROOT /opt/caffe/external/mlsl/1_mlsl_2018.0.003
ENV I_MPI_ROOT ${MLS_ROOT}
ENV PYCAFFE_ROOT $CAFFE_ROOT/python
ENV PYTHONPATH
  $(MLS_ROOT)/intel64/include:$PYCAFFE_ROOT:$PYTHONPATH
```
5.2 Network System

5.2.1 Network Overview

In the reference environment, the management interface is separated from the data interface to avoid bandwidth competition. In this environment, the management network is 10G Ethernet used for Kubernetes API interactions. The data network is Intel® OPA, with maximum bandwidth up to 100G.
As shown in the following figure, the Kubernetes network uses CNI interfaces, which can dock with different network plug-ins. Calico is a pure three-tier network connection scheme that uses the Linux kernel's forwarding engine to connect networks between containers by creating vRouter on each machine. Compared with traditional two-tier network solutions such as Flannel, Calico reduces packing and unpacking of packets, and enables more efficient transmission.

Figure 10. Network Deployment Architecture in Kubernetes*

```
5.2.2 Deploying Calico Network in Kubernetes

This section describes how we set up the Calico network for the reference design.

1. Configure RBAC required by Calico:
   kubectl apply -f \
   https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/rbac.yaml

2. Download Calico configuration:
   curl \%
   https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/hosted/calico.yaml \-O

3. Set address of etcd to ConfigMap->calico-config->etcd_endpoints:
   etcd_endpoints: http://172.16.4.19:2379

4. Set network used by Calico:
   - name: IP
     value: "autodetect"
   - name: IP_AUTODETECTION_METHOD
     value: "Can-reach=192.168.200.49"
```
5. Start network plug-ins:
   
kubectl apply -f calico.yaml

5.3 Storage Systems

5.3.1 Ceph Architecture Overview

One of the most popular distributed storage systems, Ceph has excellent performance, reliability, and scalability. It provides block storage, object storage, and file storage to meet different needs. Ceph components include Monitor, metadata software (MDS), and object storage devices (OSD), as shown in the following figure.

Figure 11. Ceph Component Architecture

Of these components, Monitor maintains various maps of cluster status, including monitor map, OSD map, placement group (PG) map and CRUSH map. MDS stores metadata for the Ceph filesystem. That means neither block storage nor object storage uses MDS. OSD is used to store data, process data replication, restore, recovery and re-balancing, and provide Monitor with monitoring information by checking the heartbeats of other OSDs.

Before data is stored, it is first divided into individual objects (default size is 4M), and then each object is stored into a unique PG using the CRUSH algorithm, with each PG storing data on its corresponding OSD. In this way, data is dispersed onto OSDs.
The Ceph distributed storage system is suitable for large-scale parallel IO read-write because of the following factors:

1. There is no center controlling node.
2. Data is dispersed onto OSDs.
3. Address mapping is implemented by the CRUSH algorithm, and clients find corresponding OSD nodes through calculation and communicate directly.

Ceph is also a distributed storage system with following features:

- High scalability: using regular x86 server to support 10-1000 servers; supporting TB to PB level expansions.
- High reliability: no single point failure, multiple copies of data, automatic management, and automatic repair.
- High performance: balanced data distribution and high degree of parallelism.

In terms of performance, Ceph provides caching mechanisms at different component levels. Reading IO does not need to consider data losses, maximizing the cache functionality.

(1) The RADOS Block Device (RBD) image has two usage modes on the client side: one mode maps to the client host through the RBD kernel driver, and the other mode is direct access through the librbd library. The former mode can use the Linux page cache, but the latter cannot. Therefore, Ceph specifically implements RBD caching function for librbd, providing LRU-based caching. In RBD caching mode, users can specify the size of the cache for read operations. Moreover, Ceph supports the pre-reading function and
can set the maximum size for pre-reading blocks, the number of pre-reading requests, and so on.

（2）Caching at BlueStore side. BlueStore is the default ObjectStore currently used in OSDs, responsible for locally storing user data in OSDs. BlueStore caching includes caching of metadata and object data, and users can set the total cache size, metadata ratio, KV metadata cache ratio, etc. Through setting and debugging, the improvement of IO reading performance at the BlueStore side can be maximized.

（3）SSD caching at Ceph client: the caching mechanism on SSDs implemented by Intel is provided on the Ceph client, by caching data to local SSDs to reduce access to backend data. This is equivalent to adding a new SSD layer between the remote OSD data and the client. When the cache is hit, the client reads the data directly from the SSD, greatly reducing the IO response time. The general framework of caching mechanisms implemented by Intel is as follows:
   - Libcachestore provides access to SSD.
   - Cache background process manages SSDs and is responsible for data down-flushing and elevating to SSDs.
   - Librbd/librgw hooks are responsible for interactions with upper tier.

**Figure 13. SSD Caching Architecture at Ceph Client**

### 5.3.2 Ceph Deployment

This section describes how we set up Ceph for the reference design.
5.3.2.1 Setting up Ceph

1. Configure Ceph Source:

   **Note:** In the steps below, replace `{ceph-stable-release}` with the release name, such as jewel, luminous, etc.

   For Ubuntu*:
   
   ```bash
   wget -q -O- 'https://download.ceph.com/keys/release.asc' | sudo apt-key add -
   echo deb https://download.ceph.com/debian-{ceph-stable-release}/ $(lsb_release -sc) main
   | sudo tee /etc/apt/sources.list.d/ceph.list
   apt-get update
   ```

   For CentOS*:
   
   ```bash
   cat << EOM > /etc/yum.repos.d/ceph.repo
   [ceph-noarch]
   name=Ceph noarch packages
   baseurl=https://download.ceph.com/rpm-{ceph-stable-release}/el7/noarch
   enabled=1
   gpgcheck=1
   type=rpm-md
   gpgkey=https://download.ceph.com/keys/release.asc
   EOM
   yum update -y
   ```

2. Install ceph-deploy:

   For Ubuntu*:
   
   ```bash
   apt-get install ceph-deploy
   ```

   For CentOS*:
   
   ```bash
   yum install ceph-deploy
   ```

   Verify ceph-deploy version:
   
   ```bash
   Ceph-deploy --version
   ```

   **Note:** Version 2.0.0 is required; older versions of ceph-deploy may not support some of the following commands.

3. Install ntp:

   For Ubuntu*:
   
   ```bash
   apt install ntp
   ```

   For CentOS*:
   
   ```bash
   yum install ntp ntpdate ntp-doc
   ```

4. Add ceph user name, making sure ceph user has root privileges without needing a password.

   ```bash
   ssh user@ceph-server // logging in to individual nodes
   sudo useradd -d /home/{username} -m {username} // adding ceph user name
   sudo passwd {username}
   ```
echo "\{username\} ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/\{username\}
sudo chmod 0440 /etc/sudoers.d/\{username\}

5. Modify host name:
  Hostname xxx.
  For example: hostname mon1

6. Make sure management node logs into individual Ceph nodes without password:
  ssh-keygen
  ssh-copy-id \{ceph-username\}@\{ceph-host_ip_address\}
  For example: ssh-copy-id mon1@192.168.1.1

7. ssh alias login
   Edit the ~/.ssh/config file. The alias should be consistent with user's host name,
   otherwise, when ceph node creates corresponding folder, inconsistencies may
   occur.

8. Turn off firewall.

5.3.2.2 Installing Ceph

1. Create installation directory at Ceph management node. Ceph-deploy will generate
   configuration files, log files, keys, etc. in this directory.
   mkdir my-cluster
   cd my-cluster

2. Create clusters:
   Ceph-deploy new \{mon_node_1\} \{mon_node_2\} \{mon_node_n\}

3. Modify newly-generated ceph.conf file based on needs.

4. Install ceph package:
   ceph-deploy install \{mon1-node\} \{mon2-node\} \{osd1-node\} \{osd2\}
   -release ceph-version

   We used the luminous release, which can only be installed using ceph-deploy
   version 2.0.0.

5. Start mon node:
   ceph-deploy mon create-initial

   You can log in into a ceph node to view current status of ceph using ceph --s
   At this time, the status of ceph is health-ok, but the cluster does not have osd
   nodes.

6. Distribute keys to individual nodes:
   Ceph-deploy admin \{mon-node\} \{osd-node\}

7. Activate management process (required for luminous release):
   ceph-deploy mgr create \{mon-node\}

   After the successful execution of this command, run ceph --s to verify active mgr.

8. Activate osd:
   ceph-deploy osd create -data /dev/{disk} \{osd-node\}
9. Verify cluster status:
   ceph-health

5.3.3 GlusterFS* Architecture Overview

GlusterFS is a scalable network storage system suitable for data-intensive tasks such as cloud storage and media streaming. GlusterFS features include extremely simple management and maintenance, unified storage for block/file/object, and support for IP/RDMA transmission protocol.

GlusterFS services provided externally are a series of logical volumes, and logical volumes in GlusterFS have a collection of storage bricks. GlusterFS can support multiple types of logical volumes to achieve different levels of data protection and access performance. These volumes include distributed storage volumes, mirrored storage volumes, distributed mirrored storage volumes, striped storage volumes, and others.

Considering the required data security for the reference design, we adopted mirrored storage volumes, with all data being replicated in all bricks consisting the volume, as shown in the following figure.

**Figure 14. Replicated Volume Mode of GlusterFS**

In this mode, the specified number of backup copies of the data are stored in different bricks. Mirrored storage can effectively prevent data losses caused by damage to the storage bricks.
5.3.4 **GlusterFS* Deployment**

This section describes how we set up GlusterFS for the reference design.

The roles of all nodes throughout the GlusterFS clusters are the same, and they all require the same software deployment. The installation method is as follows:

1. We installed release 3.12 of GlusterFS software. Since this is the latest release, conflicts may happen between some installation packages and the system itself during installation. As such, dependencies must be resolved first.
   
   ```
   rpm -e --nodeps userspace-rcu
   ```

2. Install GlusterFS clusters:
   
   ```
   yum install -y glusterfs glusterfs-server glusterfs-fuse
glusterfs-rdma glusterfs-geo-replication glusterfs-devel
   ```

3. Start service and configure automatic startup upon power on:
   
   ```
   systemctl start glusterd.service
   systemctl enable glusterd.service
   ```

4. Configure host name:
   
   ```
   /etc/hosts
   Host name pending
   ```

5. Add storage host into trusted storage pool:
   
   ```
   gluster peer probe hostname
   ```

6. View cluster status:
   
   ```
   gluster peer status
   ```

5.3.5 **GlusterFS* on Kubernetes* **

Two aspects in the application of GlusterFS in AI learning and training platform are:

1. Data is shared among the different pods in the training cluster. Data includes the sample data required for training, intermediate data used for training visualization (TensorBoard*), and the resulting model obtained from the completed training.

2. Long-term saving of user data.

The platform references GlusterFS mainly through pv and pvc, and it also needs endpoint support. In the process, endpoint is used for abstracting GlusterFS storage volumes, and then exposing the resource interface to the pod via pv. When the pod needs storage resources, it will initiate a request through pvc, and the pv will provide the user with storage resources. Some of the resources are defined as follows:

Endpoint - used primarily for abstracting storage resources:

```json
{
  "kind": "Endpoints",
  "apiVersion": "v1",
  "metadata": {
    "name": "glusterfs-cluster"
  },
  "subsets": [
  
```
"addresses": [ 
  { 
    "ip": "20.30.40.9"
  }
],
"ports": [ 
  { 
    "port": 1990
  }
]
}

Persistent volume - provides resources of storage interface for pod, and needs to reference endpoint in its definition:

apiVersion: v1
class: PersistentVolume
metadata: 
  name: gluster-dev-volume
spec: 
  capacity: 
    storage: 8Gi
  accessModes: 
    - ReadWriteMany
  glusterfs: 
    endpoints: "glusterfs-cluster"
    path: "afr-volume"
  readOnly: false

Persistent volume claim - pod initiates requests through this resource, and the system will allocate required pv resources to the pod:

kind: PersistentVolumeClaim
apiVersion: v1
metadata: 
  name: glusterfs-nginx
spec: 
  accessModes: 
    - ReadWriteMany
  resources: 
    requests: 
      storage: 8Gi

The relationship between these resources is shown in the following figure.
5.4 AI Frameworks

This section describes the AI frameworks used in the reference design.

5.4.1 Running Intel® Optimization for Caffe* on Kubernetes*

The Intel® Optimization for Caffe* MPI Master, as PS, is deployed using Kubernetes job, so that the master job exits automatically upon the completion of the distributed tasks and returns to the upper level call. The upper level calling program cleans up the MPI slave resources based on the returned results.

The MPI slave is deployed with Statefulset+Headless Service. Headless Service can provide a stable and unique network identifier that can be used to discover other members within the cluster. For example, suppose the name of StatefulSet is MPISlave, then the first initiated Pod is called MPISlave-0, the second called MPISlave-1, and so on. Headless Service can also provide stable persistent storage, orderly resource deletion, and operation termination.
In KubeDNS, the domain name resolution of Service Name directly corresponds to PodIp, without service VIP layer, so that it does not depend on kube-proxy to create iptables rules. Performance increases with the omission of iptables in kube-proxy.

**Figure 16. Intel® Optimization for Caffe* on Kubernetes**

For the detailed configuration of the reference design using MNIST as an example, refer to: https://github.com/yuntongjin/AICloudonIntelPlatform/blob/master/caffe.yaml

### 5.4.2 TensorFlow* on Kubernetes*

TensorFlow on Kubernetes uses a similar deployment method as Intel® Optimization for Caffe*: the TensorFlow session is deployed through a Kubernetes job. As such, the upper level call can immediately obtain the running state of the user program.

TensorFlow's worker and ps both use StatefulSet+Headless Service to deploy. TensorFlow must create clusters via ip, and the ip of regular pod in Kubernetes will be re-allocated with pod's rebooting. The StatefulSet+Headless Service method allows us to access a pod in a fixed way by using service_name.pod_name.

TensorBoard is deployed with the replicaset method. TensorBoard is a relatively standalone server, and we only need to consider its high availability.
Figure 17. TensorFlow* on Kubernetes*

For the detailed configuration of the reference design using MNIST as an example, refer to:
https://github.com/yuntongjin/AICloudonIntelPlatform/blob/master/tensorflow.yaml
6.0 Intel AI Edge Devices and Applications

6.1 Intel® Movidius™ NCS Overview

The built-in Myriad 2 VPU of Intel® Movidius™ Neural Compute Stick provides powerful and efficient performance to run real-time deep neural networks (DNN) directly on devices, with the performance of more than 100 billion floating-point operations per second at 1 Watt of power. This allows offline deployment for all types of AI applications. It is the world’s first USB mode-based deep learning inference tool and standalone AI accelerator. It features low power and USB interfaces and can be used for prototyping, debugging, validating, and deploying AI networks.

Figure 18. Myriad 2 Chip Structure

As the figure shows, the chip is equipped with two lightweight CPUs running a real-time system RTOS, managing various peripherals, and reading convolutional neural network models. The core component of the chip is the Vector Computing Unit (SHAVE). A chip can have up to 12 SHAVEs. The number of SHAVEs involved in calculation directly affects program performance. The Hardware Accelerators Image Signal Processing block comprises a few accelerators dedicated for image processing such as sharpening, de-noising, zooming, etc. Such accelerators are not yet available in the current development tools and will be open for support in future releases.
6.1.1 Installation

1. Prerequisites

Gather the following items:
- Intel® Movidius™ Neural Computing Stick
- Intel® Movidius™ Neural Computing SDK
- x86-64 with Ubuntu* (64 bit) 16.04 Desktop
- Raspberry Pi* 3 with Raspbian Stretch
- Network connection (with normal access to github)

2. Installation

From a terminal, change directory to the system path ready for SDK installation, and execute the below command to download the latest SDK:
```
git clone http://github.com/Movidius/ncsdk
```

Change directory to the SDK directory:
```
cd ncsdk
```

Execute installation command:
```
make install
```

After successful installation, insert the compute stick via a USB interface and execute the following command to compile "example":
```
make examples
```

3. Files that are changed after SDK installation:

```
/usr/local/include/mvnc.h
/usr/local/include/mvnc_deprecated.h
/usr/local/lib/libmvnc.so
/usr/local/lib/libmvnc.so.0
/usr/local/lib/mvnc/libmvnc.so
/usr/local/lib/mvnc/MvNCAPI.mvcmd
/usr/local/bin/mvNCCheck
/usr/local/bin/mvNCProfile
/usr/local/bin/mvNCC Compile
```
4. Uninstallation

To uninstall the SDK, from a terminal, change directory to the SDK directory and execute:

```
make uninstall
```

6.1.2 **Intel® Movidius™ Neural Compute SDK Tools**

The Intel® Movidius™ Neural Compute SDK provides tools for profiling, tuning, and compiling a deep neural network (DNN) model on a development computer (host system).

- **mvNCCompile** converts a Caffe/TensorFlow* network and associated weights to an internal Intel® Movidius™ compiled format for use with the Intel® Movidius™ Neural Compute API.
- **mvNCProfile** provides layer-by-layer statistics to evaluate the performance of Caffe/TensorFlow networks on your neural compute device.
- **mvNCCheck** compares the inference results from running the network on your neural compute device vs. Caffe/TensorFlow for network compilation validation.

**mvNCProfile**

1. **Input and Output**

A command line tool that outputs reports in html/graphics/text format.

Value "weights" of network model is not a required argument.

2. **Usage**

**Caffe syntax:**

```
mvNCProfile network.prototxt [-w weights_file] [-s Max Number of Shaves] [-in Input Node Name] [-on Output Node Name] [-is Input-Width Input-Height] [-o Output Graph Filename]
```

**Example:**

```
mvNCProfile deploy.prototxt -w bvlc_googlenet.caffemodel -s 12 -in input -on prob -is 224 224 -o GoogLeNet.graph
```

**TensorFlow syntax:**

```
mvNCProfile network.meta [-s Max Number of Shaves] [-in Input Node Name] [-on Output Node Name] [-is Input-Width Input-Height] [-o Output Graph Filename]
```

**Example:**

```
mvNCProfile inception_v1.meta -s 12 -in=input -on=InceptionV1/Logits/Predictions/Reshape_1 -is 224 224 -o InceptionV1.graph
```

**Note:** Argument `-w` can be omitted when the prefix of weights filename is the same as that of the model filename.
6.1.3 Neural Compute API (NCAPI)

Applications for performing inferences with the Intel® Movidius™ Neural Compute SDK (Intel® Movidius™ NCSDK) can be developed in either C/C++ or Python. The Neural Compute API (NCAPI) provides a software interface to load network graphs and run inferences on neural compute devices.

1. C API

The C API allows developers to call NCS hardware using C or C++ to accelerate deep neural networks. C API is provided by a header file (mvnc.h) and an associated library file (libmvnc.so). Both files are placed into the appropriate paths during SDK installation. The specific enumeration variables and function documents contained in C API can be found in this manual: https://movidius.github.io/ncsdk/ncapi/ncapi2/c_api/readme.html

2. Python API

The Python API provides Python3 API, which allows developers to use Python3 to call NCS hardware to accelerate deep neural networks. Python API is provided with a Python script (mvncapi.py). The specific enumeration variables and function documents contained in Python API can be found in this manual: https://movidius.github.io/ncsdk/ncapi/ncapi2/py_api/readme.html

More questions and related discussions can be found in this forum: https://ncsforum.movidius.com

6.1.4 Intel® Movidius™ NCSDK Examples

1. Caffe networks included in SDK Examples:
   - GoogLeNet
   - AlexNet
   - SqueezeNet

2. TensorFlow networks included in SDK Examples:
   - Inception V1
   - Inception V3

   **Note:** NCS has integrated a number of networks, such as GoogLeNet, AlexNet, SqueezeNet, etc. Developers can also use their own trained network models, if these network models are compatible with the network layers supported by NCS.

   For specific network layers supported by NCS, refer to:

3. Examples included in SDK Examples
   - hello_ncs_py // turning on or off device with Python
6.1.5 Use Case: Object Detection and Classification with Intel® Movidius™ NCS

The use case operates on Caffe and SSD MobileNet networks and accelerates with Intel® Movidius™ NCS to process camera video data in real time. Vehicles can be recognized in real time.

Deployment configuration:

- Intel® Movidius™ Neural Compute Stick
- x86_64 computer running Ubuntu® 16.04
- Intel® Optimization for Caffe*

Figure 20. Object Detection and Classification with Intel® Movidius™ NCS
Appendix A References

Intel® Xeon® Scalable Processors Artificial Intelligence Benchmarks

Product Brief: Intel® Xeon® Scalable Platform

Intel® Omni-Path White Paper

Intel® Omni-Path Software Installation Guide

RDMA
http://www.rdmamojo.com/2015/01/24/verify-rdma-working/

Deep Learning
https://blog.netapp.com/accelerate-i-o-for-your-deep-learning-pipeline/

Intel® Optimization for Caffe*
https://software.intel.com/sites/default/files/managed/bd/7d/Caffe_optimized_for_IA.pdf

Intel® Optimization for Caffe* Multinode
https://github.com/intel/caffe/wiki/Multinode-guide

How to build and run Intel Caffe Docker image
https://github.com/intel/caffe/wiki/How-to-build-run-Intel-Caffe-Docker-image

Intel® Machine Learning Scaling Library
https://github.com/intel/MLSL
TensorFlow* Optimizations
https://software.intel.com/zh-cn/articles/tensorflow-optimizations-on-modern-intel-architecture

Intel® Movidius™ Neural Compute Stick (NCS) Forum
https://ncsforum.movidius.com

Caffe* example
https://github.com/yuntongjin/AICloudonIntelPlatform/blob/master/cafe.yaml

TensorFlow* example:
https://github.com/yuntongjin/AICloudonIntelPlatform/blob/master/tensorflow.yaml

§