SX-Aurora TSUBASA for AI/ML or Non-Traditional HPC
Vector Supercomputing power in Server Chassis
Effective in AI fields requiring high memory performance for massive data processing such as demand forecasting or recommendation.
Outlook of Aurora AI Platform

- Aurora has VE (Vector CPU) and VH (IA Server: x86 CPU)
- All Framework can run on VH
- Vector preferable module can be accelerated on VE using vector ported framework

![Diagram showing the structure of Aurora AI Platform](image-url)

- **DataFrame+ML**
  - SPARK
  - Frovedis
    - MLib DataFrame
    - DataFrame
  - Scikit Learn (Python)

- **Framework**
  - TensorFlow
  - NEC TensorFlow
  - DL4J
  - DL4J -V

- **DL**

- **vector Host**
  - (IA Server: Xeon, EPYC)

- **Vector Engine**
  - (PCI Card: Vector CPU)
  - (PCI Card: Vector CPU)
Frovedis:
Framework of vectorized and distributed data analytics
Frovedis: FRamework Of VEctorized and DIStributed data analytics

- C++ framework similar to Spark
  - Supports Spark/Python interface
- MPI is used for high performance communication
- Optimized for SX-Aurora TSUBASA (also works on x86)

Open Source! [github.com/frovedis](https://github.com/frovedis)
Software for Machine Learning and AI or Frovedis Engine

- Frovedis provides wide variety of Machine Learning functions
  - It is fully tuned for Aurora Architecture

**Machine Learning**
- Linear Regression
- Ridge Regression
- Lasso Regression
- Logistic Regression
- Linear SVM
- SVD
- K-means
- Collaborative Filtering (ALS)
- word2vec
- EVD
- Factorization Machines
- Decision Trees
- Logistic Regression (Multiple Class)
- PCA
- Naïve Bayes
- Art2
- LDA
- Association Rule Mining
- Isotonic Regression
- Gaussian Mixture
- Random Forest
- GBDT
- Frequent Pattern Mining
- Spectral Clustering
- Hierarchical Clustering
- Latent Dirichlet Allocation
- Deep Learning (MLP, CNN)

**Preprocess**
- Association Rule Mining
- Page Rank
- Connected Components
- Shortest Path
- Triangle Counting

**Graph**
- Power Iteration Clustering
Machine Learning Library

Supported algorithms:

- Linear model
  - Logistic Regression
  - Multinominal Logistic Regression
  - Linear Regression
  - Linear SVM
- ALS
- K-means
- Preprocessing
  - SVD, PCA

- Word2vec
- Factorization Machines
- Decision Tree
- Naïve Bayes
- Graph algorithms
  - Shortest Path, PageRank, Connected Components

Under development:

- Frequent Pattern Mining
- Spectral Clustering
- Hierarchical Clustering
- Latent Dirichlet Allocation
- Deep Learning (MLP, CNN)
- Random Forest
- Gradient Boosting Decision Tree

We will support more!
**Frovedis Core**

- Provides Spark core-like functionalities (e.g. map, reduce)
  - Internally uses MPI to implement distributed processing
  - Inherently supports multiple cards/servers
- Users need not be aware of MPI to write distributed processing code
  - Write functions in C++
  - Provide functions to the framework to run them in parallel
- Example: double each element of distributed variable

```cpp
int two_times(int i) {return i * 2;}
int main(...) {
  ...
  dvector<int> r = d1.map(two_times);
}
```
Scatter a vector; double each element; then gather

```cpp
#include <frovedis.hpp>
using namespace frovedis;

int two_times(int i) {return i*2;}

int main(int argc, char* argv[]) {
    use_frovedis use(argc, argv);

    std::vector<int> v = {1,2,3,4,5,6,7,8};
    dvector<int> d1 = make_dvector_scatter(v);
    dvector<int> d2 = d1.map(two_times);
    std::vector<int> r = d2.gather();
}
```

Do not have to be aware of MPI (SPMD programming style)

- Looks more like a sequential program
Works as an MPI program

```cpp
#include <frovedis.hpp>
using namespace frovedis;

int two_times(int i) {return i*2;}

int main(int argc, char* argv[]) {
  use_frovedis use(argc, argv);

  std::vector<int> v = {1,2,3,4,5,6,7,8};
  dvector<int> d1 = make_dvector_scatter(v);
  dvector<int> d2 = d1.map(two_times);
  std::vector<int> r = d2.gather();
}
```

MPI_Init is called in the constructor, then branch:
- rank 0: execute the below statements
- rank 1-N: wait for RPC request from rank 0

in the destructor of “use”, MPI_Finalize is called and send RPC request to rank 1-N to stop the program

rank 0 sends RPC request to rank 1-N to do the work
Matrix Library

- Implemented using Frovedis core and existing MPI libraries[*]
  [*] ScaLAPACK/PBLAS, LAPACK/BLAS, Parallel ARPACK

- Supports dense and sparse matrix of various formats
  - Dense: row-major, column-major, block-cyclic
  - Sparse: CRS, CCS, ELL, JDS, JDS/CRS Hybrid (for better vectorization)

- Provides basic matrix operations and linear algebra
  - Dense: matrix multiply, solve, transpose, etc.
  - Sparse: matrix-vector multiply (SpMV), transpose, etc.

**Example**

```cpp
blockcyclic_matrix<double> A = X * Y; // mat mul
gesv(A, b);  // solve Ax = b
```
DataFrame

- Supports similar interface as Spark DataFrame
  - Select, Filter, Sort, Join, Group by/Aggregate
  - (SQL interface is not supported yet)

- Implemented as distributed column store
  - Each column is represented as distributed vector
  - Each operation only scans argument columns: other columns are created when necessary (late materialization)
  - Reduces size of data to access
Writing C++ programs is sometimes tedious, so we created a wrapper interface to Spark

- Call the framework through the same Spark API
- Users do not have to be aware of vector hardware

Implementation: created a server with the functionalities

- Receives RPC request from Spark and executes ML algorithm, etc.
- Only pre-built algorithms can be used from Spark

Other languages can also be supported by this architecture

- Currently Python is supported (scikit-learn API)
How it works

- Rank 0 of the Frovedis server waits for RPC from driver of Spark
- Data communication is done in parallel
  - All workers/ranks send/receive data in parallel
  - Assuming that the data can fit in the memory of the Frovedis server
Frovedis is NEC middleware and supports Spark API interface and Scikit learn (Python) API interface.

Frovedis is OSS developed by NEC (https://github.com/frovedis).

Original Spark program: logistic regression

```scala
...  
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD  
...  
val model = LogisticRegressionWithSGD.train(data)  
...  
import com.nec.froedis.mllib.classification.LogisticRegressionWithSGD  
...  
FrovedisServer.initialize(...)  
val model = LogisticRegressionWithSGD.train(data)  
FrovedisServer.shut_down()  
...```

Change to call NEC middleware implementation

```scala
...  
import com.nec.froedis.mllib.classification.LogisticRegressionWithSGD  
...  
FrovedisServer.initialize(...)  
val model = LogisticRegressionWithSGD.train(data)  
FrovedisServer.shut_down()  
...```

Only 3 line needs to be changed

Only 3 line needs to be changed

Change import

Start/Stop server

Same API (no change)
Change three lines when using Frovedis as follows

```python
from frovedis.exrpc.server import FrovedisServer # frovedis
# Change import library original scikit-learn to Frovedis
from frovedis.mllib.linear_model import LogisticRegression # frovedis
#from sklearn.linear_model import LogisticRegression # sklearn
...

# If you use Frovedis library, booting “Frovedis Server” first.
FrovedisServer.initialize("mpirun -np 4 "{os.environ['FROVEDIS_SERVER']}")

# Use the same as “scikit-learn” as follows.
clf = LogisticRegression(random_state=0, C=10.0, max_iter=10000).fit(X, y)
score = 1.0 * sum(y == y_pred) / len(y)

# Shutdown “Frovedis Server” when you finish the program.
FrovedisServer.shut_down()
```
YARN Support

Resource allocation by YARN is also supported

- Implemented in the collaboration with Cloudera (formerly Hortonworks) team

Implementation:

- YARN is modified to support Vector Engine (VE) as resource (like GPU)
- Created a wrapper program of mpirun, which works as YARN client
  - Obtain VE from YARN Resource Manager, and run MPI program on the given VE
- Used the wrapper as the server invocation command
  - Specified in FrovedisServer.initialize(...)

```plaintext
Spark ➔ mpirun wrapper ➔ YARN RM ➔ mpirun ➔ VE
```

© NEC Corporation 2020
Performance of Frovedis

- Frovedis + VE shows over 100x performance compared to Spark + x86

**Speedup (spark/x86=1)**

- Logistic Regression (web ad): Frovedis + Aurora (113.2x)
- K-means (document clustering): Spark + x86 (42.8x)
- Singular value decomposition (recommendation): Spark + x86 (56.8x)

- Spark + x86: Intel Xeon Gold 6126 x1 socket
- Aurora: Vector Engine Type 10-B (1.4GHz, 8core) x1
- Performance comparison does not include I/O time
VE with Frovedis Performance vs V100 with RAPIDS

- Frovedis is good at large and/or sparse data
  - Evaluated using such data

- Frovedis is much faster than scikit-learn
  - 10～100x speed up
  - Some scikit-learn algorithms are not parallelized

- Frovedis is much faster than RAPIDS
  - RAPIDS is GPU implementation of scikit-learn
  - Only supports limited algorithms
  - Does not support sparse data
Recommendation

- Provide recommendation service with Aurora utilizing buying/browsing history at other shop (cross recommendation)
- → Sales increased & Customer satisfaction increased

E-commerce mall site

- Shop A
  - Buy/Browse at shop A
  - Increase customer satisfaction

- Shop B
  - Recommend similar/related product
  - Buy similar/related product at Shop B
  - Increase purchase opportunity

Learning time of customer’s buying/browsing history

- Aurora: 1VE/8core
- Xeon: Gold 6126/1socket/12core

More frequent learning (e.g. daily learning → hourly learning) and enhance recommendation accuracy

22x

Increase purchase opportunity
Over 10x faster demand prediction taking into account weather, event, sales record, etc.

- **Weather forecast**
- **Event**
- **Sales data**
- **Demand prediction**
- **Inventory control manager**

**Method:** regression tree

**Aurora:** 1VE 8core
**x86:** Xeon Gold 6126 1socket 12core

10.8x
TensorFlow on SX-Aurora TSUBASA
TensorFlow now supports VE.

VE expands the convergence of TensorFlow to statistical machine learning and small model deep learning.

TensorFlow includes support for CPU, GPU, TPU, and VE.

Visit https://github.com/sx-aurora-dev/tensorflow for more information.
Deep Learning (LSTM, RNN, MLP): API’s supported in TensorFlow for VE (1/2)

<table>
<thead>
<tr>
<th>206 VE accelerated TensorFlow API’s out of 1181.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
</tr>
<tr>
<td>Acosh</td>
</tr>
<tr>
<td>Add</td>
</tr>
<tr>
<td>AddN</td>
</tr>
<tr>
<td>AddV2</td>
</tr>
<tr>
<td>ApplyAdam</td>
</tr>
<tr>
<td>ApplyGradientDescent</td>
</tr>
<tr>
<td>ApplyMomentum</td>
</tr>
<tr>
<td>ArgMax</td>
</tr>
<tr>
<td>ArgMin</td>
</tr>
<tr>
<td>Asinh</td>
</tr>
<tr>
<td>Assign</td>
</tr>
<tr>
<td>AssignAdd</td>
</tr>
<tr>
<td>AssignSub</td>
</tr>
<tr>
<td>AssignSubVariableOp</td>
</tr>
<tr>
<td>AssignVariableOp</td>
</tr>
<tr>
<td>Atanh</td>
</tr>
<tr>
<td>AvgPool</td>
</tr>
<tr>
<td>AvgPoolGrad</td>
</tr>
<tr>
<td>BiasAdd</td>
</tr>
<tr>
<td>BiasAddGrad</td>
</tr>
<tr>
<td>BroadcastGradientArgs</td>
</tr>
<tr>
<td>Case</td>
</tr>
<tr>
<td>Cast</td>
</tr>
</tbody>
</table>
Deep Learning (LSTM, RNN, MLP): API’s supported in TensorFlow for VE (2/2)

206 VE accelerated TensorFlow API’s out of 1181.

<table>
<thead>
<tr>
<th>StridedSliceGrad</th>
<th>TensorArraySizeV2</th>
<th>_ListToArray</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub</td>
<td>TensorArraySizeV3</td>
<td>_ReadVariablesOp</td>
</tr>
<tr>
<td>Sum</td>
<td>TensorArrayUnpack</td>
<td>_Retval</td>
</tr>
<tr>
<td>Switch</td>
<td>TensorArrayV2</td>
<td>_SwitchN</td>
</tr>
<tr>
<td>SymbolicGradient</td>
<td>TensorArrayV3</td>
<td>_VarHandlesOp</td>
</tr>
<tr>
<td>Tan</td>
<td>TensorArrayWrite</td>
<td>_While</td>
</tr>
<tr>
<td>TanhGrad</td>
<td>TensorArrayWriteV2</td>
<td></td>
</tr>
<tr>
<td>TemporaryVariable</td>
<td>Tile</td>
<td></td>
</tr>
<tr>
<td>TensorArray</td>
<td>Transpose</td>
<td></td>
</tr>
<tr>
<td>TensorArrayGather</td>
<td>Unpack</td>
<td></td>
</tr>
<tr>
<td>TensorArrayGatherV2</td>
<td>UnsortedSegmentSum</td>
<td></td>
</tr>
<tr>
<td>TensorArrayGatherV3</td>
<td>VarHandleOp</td>
<td></td>
</tr>
<tr>
<td>TensorArrayGrad</td>
<td>VarIsInitializedOp</td>
<td></td>
</tr>
<tr>
<td>TensorArrayGradV2</td>
<td>Variable</td>
<td></td>
</tr>
<tr>
<td>TensorArrayGradV3</td>
<td>VariableShape</td>
<td></td>
</tr>
<tr>
<td>TensorArrayGradWithShape</td>
<td>VariableV2</td>
<td></td>
</tr>
<tr>
<td>TensorArrayPack</td>
<td>While</td>
<td></td>
</tr>
<tr>
<td>TensorArrayRead</td>
<td>ZerosLike</td>
<td></td>
</tr>
<tr>
<td>TensorArrayReadV2</td>
<td>_Arg</td>
<td></td>
</tr>
<tr>
<td>TensorArrayReadV3</td>
<td>_ArrayToList</td>
<td></td>
</tr>
<tr>
<td>TensorArrayScatter</td>
<td>_DeviceArg</td>
<td></td>
</tr>
<tr>
<td>TensorArrayScatterV2</td>
<td>_DeviceRetval</td>
<td></td>
</tr>
<tr>
<td>TensorArrayScatterV3</td>
<td>_FusedBatchNormEx</td>
<td></td>
</tr>
<tr>
<td>TensorArraySize</td>
<td>_If</td>
<td></td>
</tr>
</tbody>
</table>
Vector Engine is added to a device layer

TensorFlow Architecture


And NEC is working to implement optimized kernels for VE
(206/1181 kernels are supported now)
How to use Vector Engine in a TF program

1) When a program is written in device-free manner

```python
# Creates a graph.
a = tf.constant(...)  
b = tf.constant(...)  
c = tf.matmul(a, b)
# Creates a session
sess = tf.Session()
# Runs the op.
print(sess.run(c))
```

VE is automatically used for kernels supported by VE

2) When a program uses manual device placement

```python
# Creates a graph.
with tf.device('/cpu:0')
    a = tf.constant(...)  
    b = tf.constant(...)  
    c = tf.matmul(a, b)
# Creates a session
sess = tf.Session()
# Runs the op.
print(sess.run(c))
```

Program has to be rewritten in device-free or to place to '/VE:0'

Manual device placement is also useful when you won’t use VE.

No difference from other device (CPU, GPU, TPU)
Click Prediction using Logistic Regression

“Display Advertising Challenge” on Kaggle
- Predict click-through rates on display ads
- [https://www.kaggle.com/c/criteo-display-ad-challenge/overview](https://www.kaggle.com/c/criteo-display-ad-challenge/overview)

Logistic regression based algorithm developed at NEC for benchmark
- Core kernel is sparse matrix-vector multiply that is known as memory bandwidth intensive

Benchmark Environment
- Platforms:
  - CPU: Xeon Gold 6126, 1.3TF, 120GB/s
  - GPU: V100(Pcie), 14TF, 900GB/s
  - VE: Type 10B, 4.3TF, 1.2TB/s
- Data Size: 1.4GB
Small Model Deep Learning

Example: MLP for text, models for embedded platforms

**Heavy Training**

- small models
- extensive model search for best model (ex. AutoML)
  - model architecture
  - hyper parameters

**Light Inference**

- preprocessed feature

Problem when training:

An accelerator such as GPU is under utilized because offload overhead is not negligible

 النفس days or weeks for training

Example: MLP for text, models for embedded platforms

• model architecture
• hyper parameters

cf. example of large model
- ResNet
- VGG (CNN for image classification)
VE with general purpose cores can support macro offload in addition to widely used kernel offload.

Kernel offload

initialize
epoch loop{
batch loop{
kernell

kernel

}

}

frequent offload

one-time offload

initialize
epoch loop{
batch loop{
kernell

kernel

}

}

CPU (host)

GPU, TPU, VE

5,120 comp cores (V100)

VE

mgmt and comp

Scalar Unit

Vector Unit

x8 core

CPU (host)
Malware Detection using Deep Learning

Training of small model for malware detection from a binary file
- Model is being used in NEC’s business with RAPID (NEC’s DL framework)
- We have ported the model to TF, and run on CPU, GPU and VE

Benchmark Environment
- Platform:
  - CPU: Xeon Gold 6126, 1.3TF, 120GB/s
  - GPU: V100(PCIe), 14TF, 900GB/s
  - VE: Type 10B, 4.3TF, 1.2TB/s
- Data Size: 760MB
AI / 3D Bin Packing Problems

- Solve optimal packing of multiple products in a box to reduce logistic cost
- Using a deep reinforcement learning algorithm

Combinational optimization

- Simulate optimal stowage of cargo based on size, strength, weight, loading/unloading order, etc
- Optimal arrangement

Aurora can reduce time almost by half
- **AI Framework**: TensorFlow
- **SX-Aurora TSUBASA**: 1VE 8core
- For comparison: GPU Server (Tesla P100)

No dedicated code modifications are necessary; all you need is to use TensorFlow on SX-Aurora TSUBASA

Elapse time (min)

<table>
<thead>
<tr>
<th></th>
<th>P100</th>
<th>VE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5:26:00</td>
<td>6:00:00</td>
<td>2:00:00</td>
</tr>
<tr>
<td>2:54:00</td>
<td>4:00:00</td>
<td>1.9x</td>
</tr>
</tbody>
</table>
Note

Github repository

- [https://github.com/sx-aurora-dev/tensorflow](https://github.com/sx-aurora-dev/tensorflow)

OSS Community building in progress:

- TF-VE only supports NCHW format.
- TF-VE only supports single VE card.
  - You can set VE_NODE_NUMBER to select the VE.
- TF-VE does not support all kernels in TF.
- Try with OSS philosophy!
Image and Video Processing on SX-Aurora TSUBASA
Six kernels from OpenCV that are widely used in image analytic applications are evaluated.

Memory-intensive kernels show better speedup vs GPU.

Easy to run OpenCV on VE because VE supports C/C++ and native execution model (no special language and offload required)

Speedup vs GPU (TITAN V) (VE Type 10B)

- 2D Filter: GPU is faster 10.3x, VE is faster 4.5x
- Integral Image: GPU is faster 2.6x, VE is faster 1.3x
- Opticalflow: GPU is faster 1.0x, VE is faster 1.0x
- Bilateral Filter: GPU is faster 1.0x, VE is faster 1.0x
- Feature Detection: GPU is faster 1.0x, VE is faster 1.0x
- Color conversion: GPU is faster 1.0x, VE is faster 1.0x
- Average: GPU is faster 3.4x, VE is faster 1.0x

3.4x in average
Real-time Medical Image Processing

- Medical image processing requires high memory bandwidth.
- Vector Engine (VE) is beneficial for medical image applications.

Performance Result of Image Processing

- Integral image creation and 2-Dimension filtering are utilized for medical image processing.
- 2 Test programs (NEC owned) are executed.
- NEC compared Aurora performance to TITAN V (Volta).

<table>
<thead>
<tr>
<th></th>
<th>Core</th>
<th>TFLOPS (float)</th>
<th>TB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>5120</td>
<td>13.8</td>
<td>0.650</td>
</tr>
<tr>
<td>NVIDIA TITAN V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VE</td>
<td>8</td>
<td>4.3</td>
<td>1.200</td>
</tr>
<tr>
<td>Type 10B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Higher Advantage against Volta Generation GPU**

- x4 faster for Integral Image Creation
- x10 faster for Filter 2D

GPU used for real-time Image Processing

NEC SX-Aurora TSUBASA focus on Real Application (actual user advantage) Performance rather than core peak performance (benchmarks).
Image Processing using NEC Vector Engine

Image Enhancement

Retinex

Image size: 960x693 (Processing time for 80 images)

Image size: 960x640 (Processing time for 80 images)
3D localization is utilized video analysis service especially for automobiles with spread of video recorders.

- Each of object detection and tracking is one of process modules for 3D localization.

CPU+GPU+VE system achieves higher performance than only CPU or CPU+GPU system for 3D localization.

### Topics: Applications → NEC Vector Engine (VE) Hybrid or Heterogeneous Computing (CPU+GPU+VE) Example of 3D localization

<table>
<thead>
<tr>
<th>Time for 1 frame</th>
<th>CPU Only</th>
<th>CPU + GPU</th>
<th>CPU + GPU + VE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obj det tracking</td>
<td>117ms</td>
<td>88ms</td>
<td>43ms</td>
</tr>
</tbody>
</table>

Object Detection: Yolo2
Tracking: Opticalflow-based algorithm

- CPU: Gold6126 x 2
- GPU: Quadro P2000
- VE: Type 10C (2 core)
Test system for 3D Localization

Each of images is processed with pipeline processing.

- Image1
  - 3D positioning measurement

- Image2
  - 2D Tracking
    - CPU

- Image3
  - Object Detection
    - GPU
    - VE

Hardware

<table>
<thead>
<tr>
<th></th>
<th>TFlops</th>
<th>TB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (Gold 6126)x2</td>
<td>1.0</td>
<td>0.13</td>
</tr>
<tr>
<td>GPU (P2000)</td>
<td>3.0</td>
<td>0.14</td>
</tr>
<tr>
<td>VE (Type-C: 8Core)</td>
<td>4.3</td>
<td>0.75</td>
</tr>
</tbody>
</table>

※ This system (GPU+VE) is out of guarantee.
Process of 3D localization

- 3D localization is composed of 3 process modules.
- The suitable platform (CPU/GPU/VE) is different per module.
- => CPU+GPU+VE system is best solution for 3D localization.

Process

- Object Detection
  - 2D tracking
    - 3D positioning measurement
  - CPU+VE can accelerate!

- GPU can accelerate!
  - A variety of image processing are implemented in two modules.
  - Ex.
    - 2D Filter
    - Integral Image
    - Optical flow
    - Fast Feature Detection
    - BGR2Lab
Main program is executed on CPU. Object detection module is off-loaded to GPU, and then the result is sent back to CPU. This data transmission is a general way for GPU architecture.

Server process is running on VE. If server process receives data from CPU, which transferred by socket, VE executes two modules of 2D Tracking and 3D positioning measurement.
NEC SX-Aurora TSUBASA Test Program
(NEC WING or Others)
PoC On-premise/Remote Testing Process

   - NEC provides NEC VE Appliance on-premise/Remote Access to the customer

2. Documentation or Onsite Support (If required)
   - NEC provides documentation for initial setup and check for customer

3. Tuning and Optimization Support for sample codes
   - NEC provides reference documentation for using NEC VE Ecosystem

4. Tuning and Optimization Support for customer codes
   - Customer runs sample codes either provided by NEC or by Customer

5. Generate and Study reports for Sample codes
   - Generate and Study reports for Sample codes

6. Customer seek technical support (Tuning/Optimizing)
6.1. Customer tries own source code and check performance (via reports)
   - Customer tries own source code and check performance (via reports)

7. Customer seek technical support (Tuning/Optimizing)
   - Customer seek technical support (Tuning/Optimizing)

8. PoC Success on Satisfaction
   - PoC Success on Satisfaction

© NEC Corporation 2020

Orchestrating a brighter world