Massive Parallel Computing in Cloud and kakao.
Before, Massive Computing
Improvement in kakao cloud

/32bit subnet network
BGP + NAMESPACE

Cloud Load Balancer:
+ ECMP + ARP Proxy

Cloud Tenant Network:
+ Multi Protocol Label Switching

Accelerated Cloud:
+ SR-IOV, + XDP/eBPF
Improvement in kakao cloud
Improvement in kakao cloud

**KEMI:**
Kakao Event Metering monitoring

**KOCOON:**
KakaO COntainer based service mONitoring
Improvement in kakao cloud

- **CMM0**
  - Legacy
  - Output: cloudTF

- **CMM1**
  - Self service Dev resource
  - Output: KRANE (openstack cloud)

- **CMM2**
  - Limited Prod resources
  - Output: KEMI (MaaS)

- **CMM3**
  - Automated Cloud
  - Output: DKOS (CaaS)

- **CMM4**
  - Integrated Service Platform
  - Output: 9RUM (C.N.P)
A lot of progress in kakao cloud

- Basic
  - Dev Zone

- Standard
  - Production Zone

- Secure
  - Secure Zone
  - Finance Zone
  - Payment Zone
  - Banking Zone (TBD)

- Service
  - Co-Company Zone

People, Process, Technology Change & Evolution

Kakao (End of 2019)
A lot of progress in kakao cloud
Changes in kakao with cloud

App Developer

k2hub
App Market place

D2hub
Docker registry

Jenkins

Dockerfile

Deployer

Sandbox Orchestrator
Data Ingester (kcocon)

9rum

Alpha Zone Orchestrator
Data Ingester (kcocon)

9rum

Production Zone Orchestrator
Data Ingester (kcocon)

Another App Developer

KEMI

App Developer

Data/Event

if (kakao) dev 2019
01 Massively Parallel Computing (MPC) Categories
02 Generations of MPC Programming
03 Cloud Native way of tackling technical issues in MPC
04 Application Case with Cloud Native MPC in kakao
Massively Parallel Computing
Massively Parallel Computing

Problem's section A
Problem's section B
Problem's section C

Problem should be discrete (linear/orthogonal).

instructions

Processor
Processor
Processor

Processors could be multi processor or multi machine connected by network
Massively Parallel Computing
Massively Parallel Computing

Network model pic.
Massively Parallel Computing type

Flynn Classical Taxonomy

- **SISD** (Single Instruction Single Data)
- **MISD** (Multiple Instructions Single Data)
- **SIMD** (Single Instruction Multiple Data)
- **MIMD** (Multiple Instructions Multiple Data)

Instruction Variety: Serial Program vs. Most Parallel Processing

Data Variety: Single vs. Multiple Data

Thread or OS

Most Parallel Processing
Massively Parallel Computing type

Nowadays, Change Instruction to Program

Program (package of instructions)

| STORE | ADD | LOAD | PREV |

| SPSD | MPSD |
| SPMD | MPMD |
Massively Parallel Computing

BTW, WHY MPC and Cloud #1?

How to compile and use **MPI-enabled TensorFlow**
Generations of MPC programming
Massively Parallel Computing type

Parallel Programming model
- Shared Memory Model
- Thread Model
  - POSIX thread
  - OpenMP
  - SHMEM
Massively Parallel Computing type

Parallel Programming model
• Distributed Memory Model
  • MPI (Message Passing Interface)
  • 1994 MPI 1.0 Released
  • ssh (or remote shell) based initialization

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[]) {
    int numtasks, rank, len, rc;
    char hostname[MPI_MAX_PROCESSOR_NAME];

    // initialize MPI
    MPI_Init(&argc, &argv);
    // get number of tasks
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    // get my rank
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // this one is obvious
    MPI_Get_processor_name(hostname, &len);
    printf("Number of tasks= %d My rank= %d Running on %s\n", numtasks, rank, hostname);
    // do some work with message passing
    MPI_Finalize(); }
```

Copy to every cluster nodes

```
mpirun –np 16 –hostfile hosts a.out
```
MPC generation 1

Parallel Programming model: MPI
Massively Parallel Computing type

Designing Parallel Programming:
- Domain Decomposition
  - Data is decomposed
  - Simple problem with Big Memory
- Functional Decomposition
  - Instruction is decomposed
    - Signal Processing
    - Workflow
    - Data pipe lining
MPC Considering factors

Communication overhead
- latency and bandwidth
- Visibility of communications
- Synchronous vs. asynchronous communications

Data Dependencies
\[ A(J) = A(J-1) + A(J+1) \]

Data sharing
- (Data/program) Send or Broadcast
- Shared FileSystem
MPC generation 2

Google said they found solution:
Google File System and MapReduce Framework (J Dean, SIGMOD, 2004)
- Share Nothing, Zero Down Time Computation Framework
- But, They do have issues with data communications
Google said they found another solution:

**Pregel** *(SIGMOD, 2010)*:
- BSP (Bulk Synchronous Processing) based Computational framework
- MapReduce is well suited for non-iterative, data parallelized problem
- communication is only done by predefined graph connections
Cloud Native way of Tackling MPC
Cloud Native way of tackling technical issues

- Cloud Native means it uses CONTAINER / CONTAINER ORCHESTRATOR
- Cloud Native means ‘Automation’
- Network Architecture or Communication Model Setup
  - Create Cluster Automatically
  - Create Communication Model Automatically
- Job Fault tolerance
  - Massive and Log running Job should have a way to handling ‘Failures’
  - Like MapReduce’s ‘Speculative Execution’
• Using k8s’s operator
• Operator is a representation of its component (Deployment, configmap, statefulset ..etc)
• Operator allows develop built-in plugin for your purpose
Network Architecture/Communication Model Setup

- MPI operator
  - Create MPI Cluster over k8s
  - setup ssh between the pod
    - create RSA key, save it config map, send to pod etc..
- copy MPI program to the pod
Job Fault tolerance

- Most notorious thing MPI
  - if one of the process down, all system goes down
  - so MPI developer always considering remedy for this like restart file, history file,
  - this creates the complexity
- In a Cloud Native world, you can use container level check pointing
  - CRIU (Checkpoint/Restart In User space)
    - www.criu.org
  - CRIU writes container status to share file system
  - with this you can restart the MPI pod
Application Case with Cloud Native MPC in kakao
Massively Parallel Computing

BTW, WHY MPC and Cloud #2?

#Core 000000, Memory 000TB, Disk 000TB

What if we can utilize all together?
MPC with cloud Phase 1

- Simple integration with job scheduler and container

```
#!/bin/bash
#PBS -l nodes=1:ppn=2
#PBS -l walltime=00:00:59
#PBS -l gpus=8
NV_GPU=${NV_GPU}
nvidia-docker run --net host -e PASSWORD=root -e USERNAME=root -e PORT=$PORT .io/dkos/nvidia-cuda-sshd:dev
```
MPC with cloud Phase 1

• The issues with Phase 1
  • Cannot effectively use resource for each process (training/inference/Develop)
  • Cannot elastically scale in/out the process
    • worker is statically assigned, process is also statically assigned
  • Cannot do the parallel computing with containers
    • container is just one of method for copying method
MPC with cloud Phase 2

- Using DKOSv3 (Container orchestrator as a service with k8s/dcos) in kakao, we can use k8s and MPI operator very easily
- setup YAML for computing and running

```yaml
apiVersion: kubeflow.org/v1alpha1
kind: MPIJob
metadata:
  name: mpijob
spec:
  replicas: 2
  processingResourceType: cpu
  template:
    spec:
      containers:
      - image: tf1.9.0
        command: ["sh"]
        args: [--allow-run-as-root, --allow-sudo, --allow-subprocess-inodes]
      - image: torch0.4.0
      - image: py3.5
      resources:
        limits:
          cpu: 4
      volumeMounts:
      - mountPath: /examples/MNIST-data
        name: datavolume
```
• Then the result comes out

![Graph showing processing time per number of processes with time and 병렬효율 (parallel efficiency) axes. The graph indicates a decrease in processing time as the number of processes increases, with 병렬효율 reaching a peak before decreasing.]
MPC with cloud Phase 3

- Simulation as a Service
- Put the model binary (AI/ML/MPC…), we give you the data and simulator

```
curl -X POST "http://karrozzeria:5000/1.0/simulation/tests" -d "{"runtimePath": "tensorflow", "model_data": "car_street"}"
```
MPC with cloud Phase 3 Issues

• Programming Model issue
  • Domain Decomposition overlapping and reinforcement Learning multiagent

• InterConnect issue
  • Container uses tunnel network for the inter pod networking (CNI)
  • Latency and throughput always performance bottle neck
  • kakao uses SR-IOV and VxLAN offloading for this purpose.

• File (Program, Data) Sharing issues
  • need API operated distributed file system
  • kakao has object file system (kage/tenth) and develop CSI plugin for this
• Q &A