Mingmin's Blog

About Java, Streaming, Distributed system, …

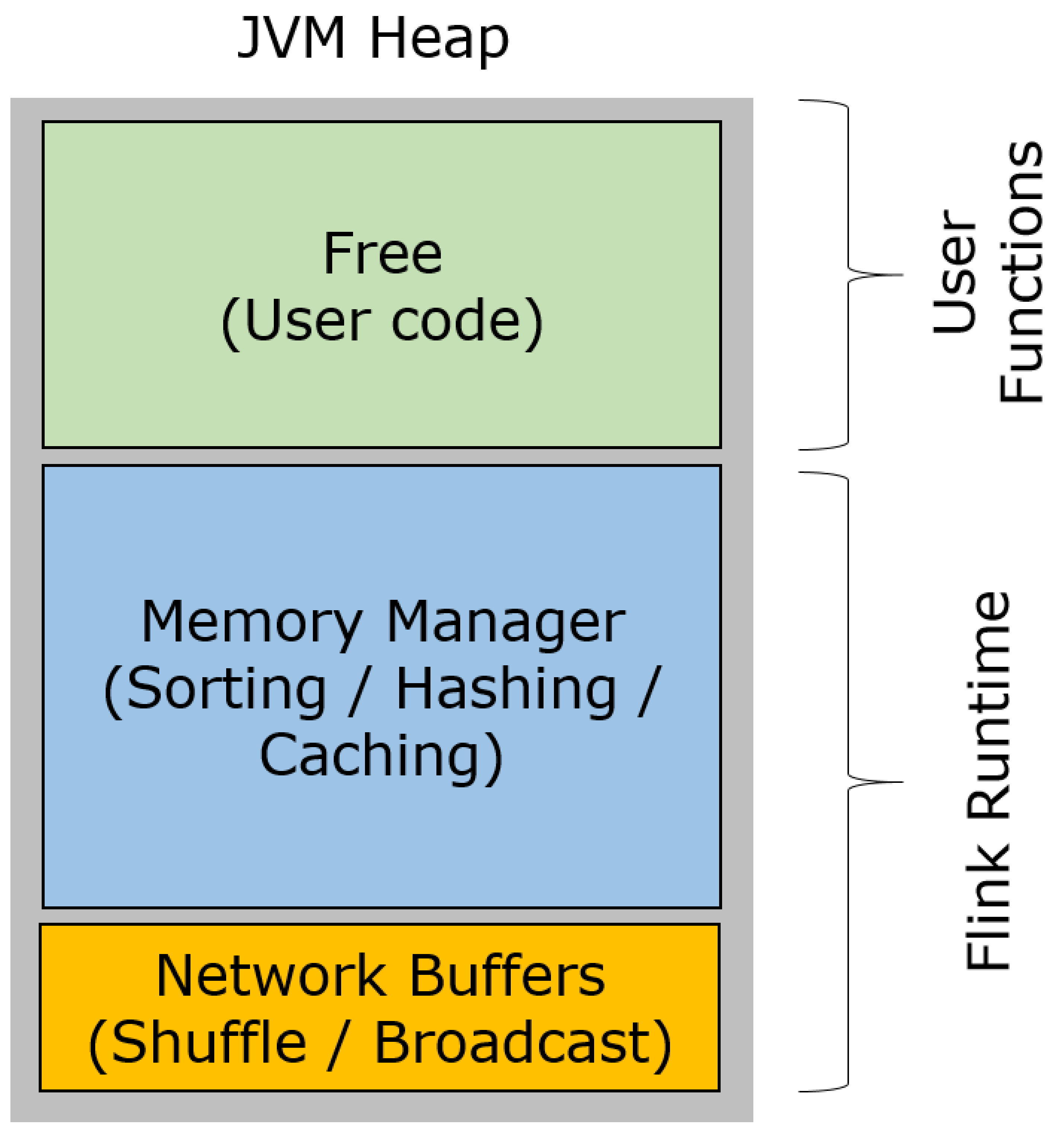
YARN

RAM allocation in Flink YARN

As YARN is very strict with killing containers which are using more memory than requested. It's critical to understand how RAM is allocated when submitting a Flink job via YARN.

In Flink[1],RAM is split into three regions:

1. Network buffers: A number of 32 KiByte buffers used by the network stack to buffer records for network transfer. Allocated on TaskManager startup. By default 2048 buffers are used, but can be adjusted via "taskmanager.network.numberOfBuffers".
2. Memory Manager pool: A large collection of buffers (32 KiBytes) that are used by all runtime algorithms whenever they need to buffer records. Records are stored in serialized form in those blocks. The memory manager allocates these buffers at startup.
3. Remaining (Free) Heap: This part of the heap is left to the user code and the TaskManager's data structures. Since those data structures are rather small, that memory is mostly available to the user code.


JVM Heap
Free (User code)
Memory Manager (Sorting / Hashing / Caching)
Network Buffers (Shuffle / Broadcast)
User Functions
Flink Runtime

Practically, `Network buffers` is set to a fixed value, **64MB** by default. More importantly, it's the balance between `Memery Manager pool` and `Remaining (Free) Heap`. The general rule is, allocate as more RAM to `Memory Managed Pool` as you can. To achieve that, let's see how RAM is allocated based on the configuration elements.

Let's dig into the detailed code:

1. JVM options in taskmanager.sh

```
# if memory allocation mode is lazy and no other JVM options are set,
# set the 'Concurrent Mark Sweep GC'
if [[ $FLINK_TM_MEM_PRE_ALLOCATE == "false" ]] && [ -z "${FLINK_ENV_JAVA_OPTS}" ] && [ -z "${FLINK_ENV_JAVA_OPTS_TM}" ]; then
    export JVM_ARGS="$JVM_ARGS -XX:+UseG1GC"
fi

if [[ ! ${FLINK_TM_HEAP} =~ ${IS_NUMBER} ]] || [[ "${FLINK_TM_HEAP}" -lt "0" ]]; then
    echo "[ERROR] Configured TaskManager JVM heap size is not a number. Please set '${KEY_TASKM_MEM_SIZE}' in ${FLINK_CONF_FILE}."
    exit 1
fi

if [ "${FLINK_TM_HEAP}" -gt "0" ]; then

    TM_HEAP_SIZE=$(calculateTaskManagerHeapSizeMB)
    # Long.MAX_VALUE in TB: This is an upper bound, much less direct memory will be used
    TM_MAX_OFFHEAP_SIZE="8388607T"

    export JVM_ARGS="${JVM_ARGS} -Xms${TM_HEAP_SIZE}M -Xmx${TM_HEAP_SIZE}M -XX:MaxDirectMemorySize=${TM_MAX_OFFHEAP_SIZE}"

fi

# Add TaskManager-specific JVM options
export FLINK_ENV_JAVA_OPTS="${FLINK_ENV_JAVA_OPTS} ${FLINK_ENV_JAVA_OPTS_TM}"
```

Notes:
1. if `taskmanager.memory.preallocate` is false(default), and no value set for `env.java.opts.taskmanager`, it adds GC option `-XX:+UseG1GC`;
2. `TM_HEAP_SIZE` is calculated in function `calculateTaskManagerHeapSizeMB`(see below for details), then the final JVM_ARGS is `"${JVM_ARGS} -Xms${TM_HEAP_SIZE}M -Xmx${TM_HEAP_SIZE}M -XX:MaxDirectMemorySize=${TM_MAX_OFFHEAP_SIZE}"`. That means all `TM_HEAP_SIZE` are allocated at the beginning, and the rest are assigned as `-XX:MaxDirectMemorySize`;

2. function `calculateTaskManagerHeapSizeMB`

```
# same as org.apache.flink.runtime.taskexecutor.TaskManagerServices.calculateHeapSizeMB(long totalJavaMemorySizeMB, Configuration config)
calculateTaskManagerHeapSizeMB() {
    if [ "${FLINK_TM_HEAP}" -le "0" ]; then
        echo "Variable 'FLINK_TM_HEAP' not set (usually read from '${KEY_TASKM_MEM_SIZE}' in ${FLINK_CONF_FILE})."
        exit 1
    fi

    local network_buffers_mb=$(($(calculateNetworkBufferMemory) >> 20)) # bytes to megabytes
    # network buffers are always off-heap and thus need to be deduced from the heap memory size
    local tm_heap_size_mb=$((${FLINK_TM_HEAP} - network_buffers_mb))

    if useOffHeapMemory; then

        if [[ "${FLINK_TM_MEM_MANAGED_SIZE}" -gt "0" ]]; then
            # We split up the total memory in heap and off-heap memory
            if [[ "${tm_heap_size_mb}" -le "${FLINK_TM_MEM_MANAGED_SIZE}" ]]; then
                echo "[ERROR] Remaining TaskManager memory size (${tm_heap_size_mb} MB, from: '${KEY_TASKM_MEM_SIZE}' (${FLINK_TM_HEAP} MB) minus network buffer memory size (${network_buffers_mb} MB, from: '${KEY_TASKM_NET_BUF_FRACTION}', '
                exit 1
            fi

            tm_heap_size_mb=$((tm_heap_size_mb - FLINK_TM_MEM_MANAGED_SIZE))
        else
            # Bash only performs integer arithmetic so floating point computation is performed using awk
            if [[ -z "${HAVE_AWK}" ]] ; then
                command -v awk >/dev/null 2>&1
                if [[ $? -ne 0 ]]; then
                    echo "[ERROR] Program 'awk' not found."
                    echo "Please install 'awk' or define '${KEY_TASKM_MEM_MANAGED_SIZE}' instead of '${KEY_TASKM_MEM_MANAGED_FRACTION}' in ${FLINK_CONF_FILE}."
                    exit 1
                fi
                HAVE_AWK=true
            fi

            # We calculate the memory using a fraction of the total memory
            if [[ `awk '{ if ($1 > 0.0 && $1 < 1.0) print "1"; }' <<< "${FLINK_TM_MEM_MANAGED_FRACTION}"` != "1" ]]; then
                echo "[ERROR] Configured TaskManager managed memory fraction '${FLINK_TM_MEM_MANAGED_FRACTION}' is not a valid value."
                echo "It must be between 0.0 and 1.0."
                echo "Please set '${KEY_TASKM_MEM_MANAGED_FRACTION}' in ${FLINK_CONF_FILE}."
                exit 1
            fi

            # recalculate the JVM heap memory by taking the off-heap ratio into account
            local offheap_managed_memory_size=`awk "BEGIN { printf \"%.0f\n\", ${tm_heap_size_mb} * ${FLINK_TM_MEM_MANAGED_FRACTION} }"`
            tm_heap_size_mb=$((tm_heap_size_mb - offheap_managed_memory_size))
        fi
    fi

    echo ${tm_heap_size_mb}
}
```

Note:
1. if `taskmanager.memory.off-heap` is false, all free memory are used in `-Xms -Xmx`;
2. if `taskmanager.memory.off-heap` is true, `heapSizeMB = (totalJavaMemorySizeMB - networkBufMB) * (1.0 - taskmanager.memory.fraction)`;
3. networkBufMB = min(taskmanager.network.memory.max, max(taskmanager.network.memory.min, totalJavaMemorySizeMB * taskmanager.network.memory.fraction));
4. free memory or `totalJavaMemorySizeMB` equals to tm_total_memory – containerized.heap-cutoff in YARN mode;

In summary, these parameters are used in YARN mode:
1. `-ytm,--yarntaskManagerMemory` Memory per TaskManager Container [in MB];
2. `taskmanager.heap.mb`: JVM heap size (in megabytes) for the TaskManagers, which are the parallel workers of the system. In contrast to Hadoop, Flink runs operators (e.g., join, aggregate) and user-defined functions (e.g., Map, Reduce, CoGroup) inside the TaskManager (including sorting/hashing/caching), so this value should be as large as possible. If the cluster is exclusively running Flink, the total amount of available memory per machine minus some memory for the operating system (maybe 1-2 GB) is a good value. On YARN setups, this value is automatically configured to the size of the TaskManager's YARN container, minus a certain tolerance value.

1. `containerized.heap-cutoff-ratio: (Default 0.25)` Percentage of heap space to remove from containers started by YARN. When a user requests a certain amount of memory for each TaskManager container (for example 4 GB), we can not pass this amount as the maximum heap space for the JVM (-Xmx argument) because the JVM is also allocating memory outside the heap. YARN is very strict with killing containers which are using more memory than requested. Therefore, we remove this fraction of the memory from the requested heap as a safety margin and add it to the memory used off-heap.
2. `containerized.heap-cutoff-min: (Default 600 MB)` Minimum amount of memory to cut off the requested heap size.

3. `taskmanager.memory.size`: The amount of memory (in megabytes) that the task manager reserves on-heap or off-heap (depending on taskmanager.memory.off-heap) for sorting, hash tables, and caching of intermediate results. If unspecified (-1), the memory manager will take a fixed ratio with respect to the size of the task manager JVM as specified by taskmanager.memory.fraction. (DEFAULT: -1)

4. `taskmanager.memory.fraction`: The relative amount of memory (with respect to taskmanager.heap.mb, after subtracting the amount of memory used by network buffers) that the task manager reserves for sorting, hash tables, and caching of intermediate results. For example, a value of 0.8 means that a task manager reserves 80% of its memory (on-heap or off-heap depending on taskmanager.memory.off-heap) for internal data buffers, leaving 20% of free memory for the task manager's heap for objects created by user-defined functions. (DEFAULT: 0.7) This parameter is only evaluated, if taskmanager.memory.size is not set.
5. `taskmanager.memory.off-heap`: If set to true, the task manager allocates memory which is used for sorting, hash tables, and caching of intermediate results outside of the JVM heap. For setups with larger quantities of memory, this can improve the efficiency of the operations performed on the memory (DEFAULT: false).

6. `taskmanager.network.memory.fraction`: Fraction of JVM memory to use for network buffers. This determines how many streaming data exchange channels a TaskManager can have at the same time and how well buffered the channels are. If a job is rejected or you get a warning that the system has not enough buffers available, increase this value or the min/max values below. (DEFAULT: 0.1)

7. `taskmanager.network.memory.min`: Minimum memory size for network buffers in bytes (DEFAULT: 64 MB)
8. `taskmanager.network.memory.max`: Maximum memory size for network buffers in bytes (DEFAULT: 1 GB)

Let's see some examples:

case 1. off-heap = false / taskmanager.memory.size = -1(default)

Input settings

```
-ytm 2048
taskmanager.memory.off-heap     false
taskmanager.memory.preallocate  true
taskmanager.memory.fraction     0.1
```

JVM options for TaskManager container:

```
2018-07-02 16:45:05.762 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Maximum heap size: 1388 MiBytes
2018-07-02 16:45:05.762 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JAVA_HOME: /usr/share/jdk1.8.0_144
2018-07-02 16:45:05.763 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Hadoop version: 2.7.3
2018-07-02 16:45:05.763 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JVM Options:
2018-07-02 16:45:05.763 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xms1448m
2018-07-02 16:45:05.763 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xmx1448m
```

It is: -Xmx1448m = 2048 – max(2048*0.25=512, 600)

case 2. off-heap = false / taskmanager.memory.size = 256

Input settings

```
-ytm 2048
taskmanager.memory.off-heap     false
taskmanager.memory.preallocate  true
taskmanager.memory.fraction     0.1
taskmanager.memory.size     256
```

JVM options for TaskManager container:

```
2018-07-02 17:34:45.339 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Maximum heap size: 1388 MiBytes
2018-07-02 17:34:45.339 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JAVA_HOME: /usr/share/jdk1.8.0_144
2018-07-02 17:34:45.339 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Hadoop version: 2.7.3
2018-07-02 17:34:45.340 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JVM Options:
2018-07-02 17:34:45.340 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xms1448m
2018-07-02 17:34:45.340 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xmx1448m
```

It is: -Xmx1448m = 2048 – max(2048*0.25=512, 600)

case 3. off-heap = true / taskmanager.memory.size = -1(default)

Input settings

```
-ytm 2048
taskmanager.memory.fraction     0.1
taskmanager.memory.off-heap     true
taskmanager.memory.preallocate  true
```

JVM options for TaskManager container:

```
2018-07-02 16:52:19.534 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Maximum heap size: 1125 MiBytes
2018-07-02 16:52:19.534 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JAVA_HOME: /usr/share/jdk1.8.0_144
2018-07-02 16:52:19.535 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Hadoop version: 2.7.3
2018-07-02 16:52:19.535 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JVM Options:
2018-07-02 16:52:19.535 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xms1174m
2018-07-02 16:52:19.535 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xmx1174m
2018-07-02 16:52:19.535 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -XX:MaxDirectMemorySize=874m
```

It is:
totalJavaMemorySizeMB = 2048 – max(2048*0.25=512, 600) = 1448*
*networkBufMB = min(1024, max(64, 1448*0.1) = 144.8
-Xmx1174m ~ (1448 – 144.8) * (1.0 – 0.1) = 1172.88

-XX:MaxDirectMemorySize = 874m ~ cutoff(max(2048*0.25=512, 600)) + netBuffer(144.8) + managedMemory((1448 – 144.8) * 0.1=130.32) = 875.12

case 4. off-heap = true / taskmanager.memory.size = 256

Input settings

```
-ytm 2048
taskmanager.memory.fraction     0.1
taskmanager.memory.off-heap     true
taskmanager.memory.preallocate  true
taskmanager.memory.size     256
```

JVM options for TaskManager container:

```
2018-07-02 17:37:21.247 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Maximum heap size: 1004 MiBytes
2018-07-02 17:37:21.247 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JAVA_HOME: /usr/share/jdk1.8.0_144
2018-07-02 17:37:21.248 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  Hadoop version: 2.7.3
2018-07-02 17:37:21.248 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -  JVM Options:
2018-07-02 17:37:21.248 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xms1048m
2018-07-02 17:37:21.248 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -Xmx1048m
2018-07-02 17:37:21.248 [main] INFO  org.apache.flink.yarn.YarnTaskManagerRunner  -     -XX:MaxDirectMemorySize=1000m
```

It is:
totalJavaMemorySizeMB = 2048 – max(2048*0.25=512, 600) = 1448*
*networkBufMB = min(1024, max(64, 1448*0.1) = 144.8
-Xmx1048m ~ (1448 – 144.8) – 256 = 1047.2

-XX:MaxDirectMemorySize = 1000 ~ cutoff(max(2048*0.25=512, 600)) + netBuffer(144.8) + managedMemory(256) = 1000.8

[1]. https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=53741525 (https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=53741525)

Posted in StreamingEco and tagged flink, memory, YARN on 2018/07/02 by Mingmin. Leave a comment

Add nodes to DFS/YARN cluster

In previous post (https://mingminxu.com/2017/11/05/setup-a-yarn-cluster-with-ha/) I created a HDFS/YARN cluster with only 4 nodes. Now I've requested more resource, it's time to extend the capacity.

Here I plan to add another 4 hosts to the cluster, both HDFS and YARN. Please node that if the host profile(mostly RAM) is not the same, you should change `yarn.nodemanager.resource.memory-mb` in `etc/hadoop/yarn-site.xml` properly.

1. environment setup

First of all, let's setup the environment for each host.

a. enable passwordless access for new added hosts;

b. copy hadoop-2.7.4.tar.gz, jdk-7u71-linux-x64.tar.gz and jdk-8u144-linux-x64.tar.gz to all hosts, and unzip them;

```
tar -zxf hadoop-2.7.4.tar.gz ;
tar -zxf jdk-7u71-linux-x64.tar.gz ;
tar -zxf jdk-8u144-linux-x64.tar.gz;
```

c. create local directories

```
mkdir -p /mnt/dfs/namenode /mnt/dfs/data /mnt/yarn/nm-local-dir /mnt/yarn/nm-log /mnt/dfs/journal;
chown -R stack:stack /mnt/dfs/ /mnt/yarn
```

d. add profile /etc/profile.d/hadoop.sh

```
HADOOP_HOME=/home/stack/hadoop-2.7.4
export HADOOP_HOME
HADOOP_PREFIX=/home/stack/hadoop-2.7.4
export HADOOP_PREFIX
export HADOOP_CONF_DIR=/home/stack/hadoop-2.7.4/etc/hadoop
export YARN_CONF_DIR=/home/stack/hadoop-2.7.4/etc/hadoop
```

2. DFS/YARN configuration

Since all my hosts have the same size, I only need to add new host list to `etc/hadoop/slaves` and copy all configurations to new hosts.

3. enable DataNode/NodeManager services

```
$HADOOP_PREFIX/sbin/hadoop-daemon.sh --config $HADOOP_CONF_DIR --script hdfs start datanode

$HADOOP_HOME/sbin/yarn-daemon.sh --config $HADOOP_HOME/etc/hadoop start nodemanager
```

Now you should see the new nodes in both HDFS and YARN cluster.

Posted in HadoopEco and tagged capacity, extend, flex up, HDFS, YARN on 2017/11/05 by Mingmin. Leave a comment

OLDER POSTS