

《大数据应用项目实践》课程设计

(2023/2024 学年第二学期)

指导教师：夏劲松、霍戍文

实践总结报告

计算机科学与技术（人工智能）学院

2024 年 05 月

浙江理工大学实践总结报告

姓名	陈昊天	性别	男	学号	2021329600006
分院	计算机科学与技术学院	专业	计算机科学与技术		
实践单位	浙江理工大学	实践岗位	大数据应用项目实践	实践时间	5月21日-6月23日

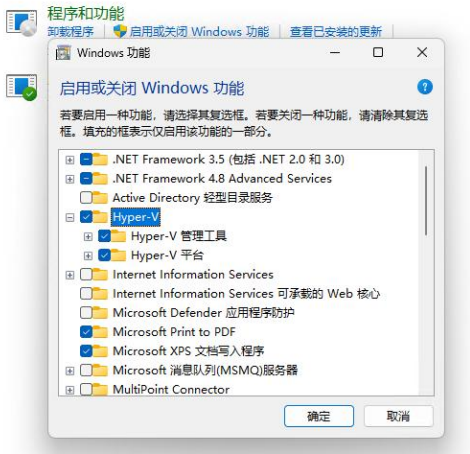
(一) 实践主要内容及进程

(1) Linux 操作系统与虚拟机

模块一 环境搭建

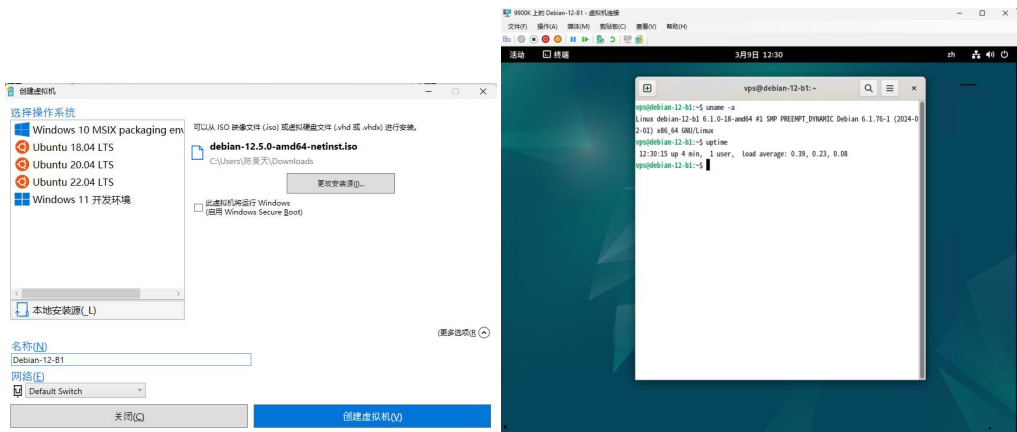
1 安装 Hyper-V

在控制面板-程序-启用或关闭 Windows 功能中启用 Hyper-V



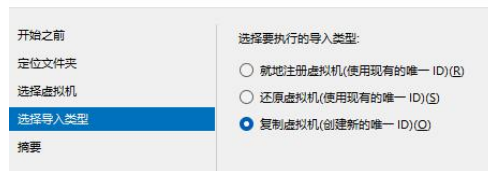
2 安装 Debian 12

在 <https://www.debian.org/download.zh-cn.html> 下载 Debian 12 镜像，并创建虚拟机。



3 复制 2 台镜像机

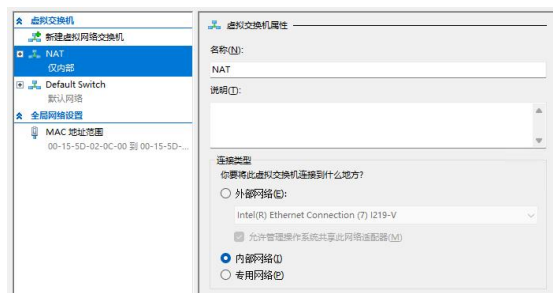
导出虚拟机，然后再导入虚拟机



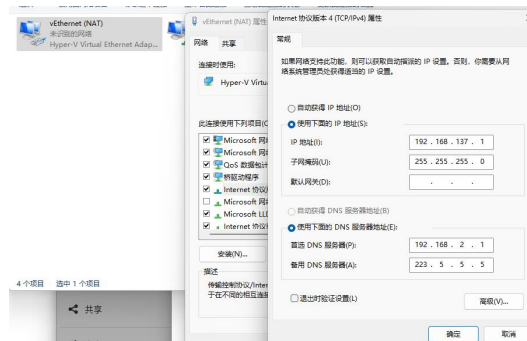
虚拟机①	
名称	状态
D12B1	关机
D12B2	关机
D12B3	关机

4 网络配置和系统管理操作

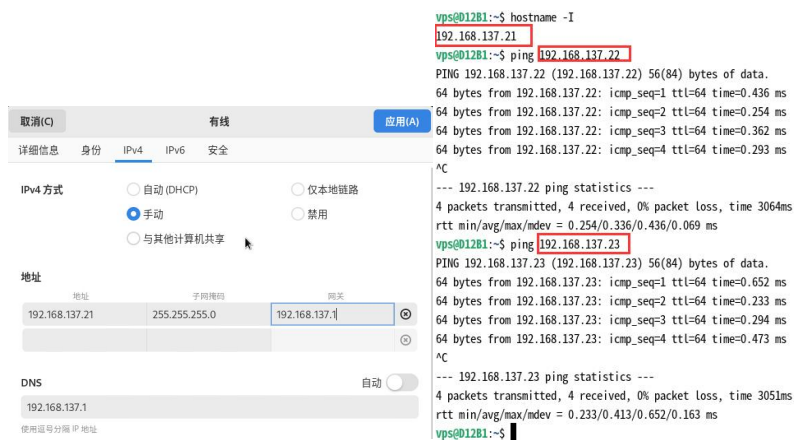
新建内部虚拟交换机 (NAT) ， 分配到 3 台虚拟机



设置虚拟交换机， 手动设置 IP 地址



在 3 台虚拟机内分别手动设置 IP 地址为 192.168.137.21, 192.168.137.22, 192.168.137.23

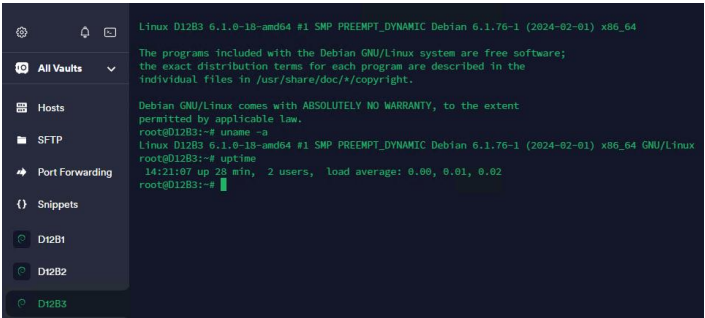


5 安装 Termius 远程登录

在虚拟机上安装 openssh-server

apt update

apt install openssh-server
修改/etc/ssh/sshd_config, 使允许 root 登录
修改 PermitRootLogin 属性为 yes, 随后重启 ssh
随后在 Termius 上填写虚拟机 IP 地址、root 密码即可远程登陆



(2) Linux 常用命令与 Shell 编程

功能	步骤	结果
分区格式化和挂载	使用 lsblk 列出所有可用的块设备	<pre>root@D12B1:~# lsblk NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS sda 8:0 0 127G 0 disk -sda1 8:1 0 512M 0 part /boot/efi -sda2 8:2 0 125.5G 0 part / `-sda3 8:3 0 976M 0 part [SWAP] sdb 8:16 0 1G 0 disk</pre>
	使用 fdisk /dev/sdb 创建分区	<pre>root@D12B1:~# fdisk /dev/sdb Welcome to fdisk (util-linux 2.38.1). Changes will remain in memory only, until you decide to write them. Be careful before using the write command. Device does not contain a recognized partition table. Created a new DOS (MBR) disklabel with disk identifier 0xec07a6c4. Command (m for help): g Created a new GPT disklabel (GUID: AA5EC750-BD2D-5A47-84BD-87304C4727C4). Command (m for help): n Partition number (1-128, default 1): First sector (2048-2097118, default 2048): Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-2097118, default 2095103): Created a new partition 1 of type 'Linux filesystem' and of size 1022 MiB.</pre>

		Command (m for help): w The partition table has been altered. Calling ioctl() to re-read partition table. Syncing disks.																																																												
使用 mkfs 格式化分区		root@D12B1:~# mkfs.ext4 /dev/sdb1 mke2fs 1.47.0 (5-Feb-2023) Discarding device blocks: done Creating filesystem with 261632 4k blocks and 65408 inodes Filesystem UUID: 801282f5-1f3b-4022-8454-1dc589827c7c Superblock backups stored on blocks: 32768, 98304, 163840, 229376 Allocating group tables: done Writing inode tables: done Creating journal (4096 blocks): done Writing superblocks and filesystem accounting information: done																																																												
使用 mount 挂载分区		root@D12B1:~# mkdir /mnt/mydisk root@D12B1:~# mount /dev/sdb1 /mnt/mydisk root@D12B1:~# df -h <table><thead><tr><th>Filesystem</th><th>Size</th><th>Used</th><th>Avail</th><th>Use%</th><th>Mounted on</th></tr></thead><tbody><tr><td>udev</td><td>919M</td><td>0</td><td>919M</td><td>0%</td><td>/dev</td></tr><tr><td>tmpfs</td><td>191M</td><td>980K</td><td>190M</td><td>1%</td><td>/run</td></tr><tr><td>/dev/sda2</td><td>124G</td><td>6.0G</td><td>111G</td><td>6%</td><td>/</td></tr><tr><td>tmpfs</td><td>951M</td><td>0</td><td>951M</td><td>0%</td><td>/dev/shm</td></tr><tr><td>tmpfs</td><td>5.0M</td><td>0</td><td>5.0M</td><td>0%</td><td>/run/lock</td></tr><tr><td>/dev/sda1</td><td>511M</td><td>5.9M</td><td>506M</td><td>2%</td><td>/boot/efi</td></tr><tr><td>tmpfs</td><td>191M</td><td>44K</td><td>190M</td><td>1%</td><td>/run/user/0</td></tr><tr><td>tmpfs</td><td>191M</td><td>60K</td><td>190M</td><td>1%</td><td>/run/user/113</td></tr><tr><td>/dev/sdb1</td><td>988M</td><td>24K</td><td>921M</td><td>1%</td><td>/mnt/mydisk</td></tr></tbody></table>	Filesystem	Size	Used	Avail	Use%	Mounted on	udev	919M	0	919M	0%	/dev	tmpfs	191M	980K	190M	1%	/run	/dev/sda2	124G	6.0G	111G	6%	/	tmpfs	951M	0	951M	0%	/dev/shm	tmpfs	5.0M	0	5.0M	0%	/run/lock	/dev/sda1	511M	5.9M	506M	2%	/boot/efi	tmpfs	191M	44K	190M	1%	/run/user/0	tmpfs	191M	60K	190M	1%	/run/user/113	/dev/sdb1	988M	24K	921M	1%	/mnt/mydisk
Filesystem	Size	Used	Avail	Use%	Mounted on																																																									
udev	919M	0	919M	0%	/dev																																																									
tmpfs	191M	980K	190M	1%	/run																																																									
/dev/sda2	124G	6.0G	111G	6%	/																																																									
tmpfs	951M	0	951M	0%	/dev/shm																																																									
tmpfs	5.0M	0	5.0M	0%	/run/lock																																																									
/dev/sda1	511M	5.9M	506M	2%	/boot/efi																																																									
tmpfs	191M	44K	190M	1%	/run/user/0																																																									
tmpfs	191M	60K	190M	1%	/run/user/113																																																									
/dev/sdb1	988M	24K	921M	1%	/mnt/mydisk																																																									

进程线程类	使用 ps aux grep xxx 查看 xxx 相关进程	<pre> root@D12B1:~# ps aux grep gnome Debian--+ 702 0.0 0.4 159644 8192 tty1 Ssl+ 14:37 0:00 /usr/libexec/gdm-wayland-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart Debian--+ 708 0.0 0.0 6260 1492 tty1 S+ 14:37 0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart Debian--+ 714 0.0 1.1 520956 22372 tty1 Sl+ 14:37 0:00 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart Debian--+ 747 0.2 12.4 5267812 242040 tty1 Sl+ 14:37 0:02 /usr/bin/gnome-shell Debian--+ 877 0.0 1.4 2793204 29104 tty1 Sl+ 14:37 0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.Notifications Debian--+ 880 0.0 0.4 164312 9444 tty1 Sl+ 14:37 0:00 /usr/libexec/at-spi2-registryd --use-gnome-session Debian--+ 1101 0.0 1.6 2727652 31480 tty1 Sl+ 14:37 0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.ScreenSaver root 1175 0.0 0.0 3324 1444 pts/0 S+ 14:56 0:00 grep gnome </pre>
	使用 pkill 终止进程	<pre> root@D12B1:~# tmux [detached (from session 0)] root@D12B1:~# ps aux grep tmux root 1181 0.0 0.2 9044 3924 ? Ss 14:58 0:00 tmux root 1189 0.0 0.0 3324 1408 pts/0 S+ 14:59 0:00 grep tmux root@D12B1:~# pkill tmux root@D12B1:~# ps aux grep tmux root 1195 0.0 0.0 3324 1488 pts/0 S+ 14:59 0:00 grep tmux </pre>
	使用 lsof -i:端口号 查看端口占用情况	<pre> root@D12B1:~# lsof -i:22 COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME sshd 612 root 3u IPv4 19913 0t0 TCP *:ssh (LISTEN) sshd 612 root 4u IPv6 19915 0t0 TCP *:ssh (LISTEN) sshd 648 root 4u IPv4 15136 0t0 TCP D12B1:ssh->9900K:62018 (ESTABLISHED) </pre>
目录和文件操	pwd 显示当前工作目录的绝对路径	<pre> root@D12B1:~# pwd /root </pre>

作	ls 列出目录的内容	<pre> root@D12B1:/# ls -a . bin etc initrd.img.old lost+found opt run sys var .. boot home lib media proc sbin tmp vmlinuz .cache dev initrd.img lib64 mnt root srv usr vmlinuz.old root@D12B1:/# </pre>
	cd 切换目录	<pre> root@D12B1:/# cd /etc root@D12B1:/etc# </pre>
	mkdir 创建一个新的目录	<pre> root@D12B1:~# mkdir test root@D12B1:~# ls test root@D12B1:~# </pre>
	rmdir 删除一个空的目录	<pre> root@D12B1:~# rmdir test root@D12B1:~# ls root@D12B1:~# </pre>
	touch 创建空文件	<pre> root@D12B1:~# touch 1.txt root@D12B1:~# ls 1.txt root@D12B1:~# </pre>
	cp 复制文件或目录	<pre> root@D12B1:~# cp 1.txt 2.txt root@D12B1:~# ls 1.txt 2.txt </pre>
	rm 移除文件或目录	<pre> root@D12B1:~# rm 1.txt root@D12B1:~# ls 2.txt </pre>
	mv 移动文件与目录或重命名	<pre> root@D12B1:~# mv 2.txt 3.txt root@D12B1:~# ls 3.txt root@D12B1:~# </pre>
	cat 查看文件内容	<pre> root@D12B1:~# echo 123 >> 3.txt root@D12B1:~# cat 3.txt 123 </pre>
	more 文件内容分屏查看器	<pre> root@D12B1:~# more 3.txt </pre>
	less 分屏显示文件内容	<pre> root@D12B1:~# less 3.txt </pre>
	echo 输出内容到控制台	<pre> root@D12B1:~# echo 123 123 </pre>
	head 显示文件头部内容	<pre> root@D12B1:~# head 3.txt 123 </pre>
	tail 输出文件尾部内容	<pre> root@D12B1:~# tail 3.txt 123 </pre>

	> 输出重定向 和 >> 追加	<pre> root@D12B1:~# echo 123456 > 3.txt root@D12B1:~# cat 3.txt 123456 root@D12B1:~# echo 666 >> 3.txt root@D12B1:~# cat 3.txt 123456 666 root@D1 </pre>
	ln 软链接	<pre> root@D12B1:~# ln -s 3.txt 4.txt root@D12B1:~# ls 3.txt 4.txt root@D12B1:~# cat 4.txt 123456 666 root@D12B1:~# </pre>
	history 查看已经执行过历史命令	<pre> root@D12B1:~# history 1 cd 2 apt install openssh-server 3 vim /etc/ssh/sshd_config 4 apt install vim 5 vim /etc/ssh/sshd_config 6 systemctl restart ssh 7 uname -a 8 uptime 9 tmux 10 apt install tmux 11 tmux </pre>
时间日期类	date 显示当前时间	<pre> root@D12B1:~# date Sat Mar 9 15:33:34 CST 2024 </pre>
	ncal 查看日历	<pre> root@D12B1:~# ncal March 2024 Su 3 10 17 24 31 Mo 4 11 18 25 Tu 5 12 19 26 We 6 13 20 27 Th 7 14 21 28 Fr 1 8 15 22 29 Sa 2 9 16 23 30 </pre>
用户管理命令	useradd 添加新用户	<pre> root@D12B1:~# useradd test </pre>
	passwd 设置用户密码	<pre> root@D12B1:~# passwd test New password: Retype new password: passwd: password updated successfully </pre>
	id 查看用户是否存在	<pre> root@D12B1:~# id test uid=1001(test) gid=1001(test) groups=1001(test) </pre>

		<pre> root@D12B1:~# cat /etc/passwd root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534:./nonexistent:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:998:998:systemd Network Management:./usr/sbin/nologin tss:x:100:107:TPM software stack,,,:/var/lib/tpm:/bin/false systemd-timesync:x:997:997:systemd Time Synchronization:./usr/sbin/nologin messagebus:x:101:108:./nonexistent:/usr/sbin/nologin usbmux:x:102:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin sshd:x:103:65534:./run/sshd:/usr/sbin/nologin dnsmasq:x:104:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin /nologin avahi:x:105:112:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin speech-dispatcher:x:106:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false fwupd-refresh:x:107:115:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin saned:x:108:117:./var/lib/saned:/usr/sbin/nologin geoclue:x:109:118:./var/lib/geoclue:/usr/sbin/nologin polkitd:x:996:996:polkit:/nonexistent:/usr/sbin/nologi n rtkit:x:110:119:RealtimeKit,,,:/proc:/usr/sbin/nologin colord:x:111:120:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin </pre>	
	cat /etc/passwd 查看创建了哪些用户		

用户组管理命令		<pre> gnome-initial-setup:x:112:65534:./run/gnome-initial-setup:/bin/false Debian-gdm:x:113:121:Gnome Display Manager:/var/lib/gdm3:/bin/false vps:x:1000:1000:vps,,,:/home/vps:/bin/bash test:x:1001:1001:./home/test:/bin/sh </pre>
	su 切换用户	<pre> root@D12B1:~# su test \$ </pre>
	userdel 删除用户	<pre> root@D12B1:~# userdel test </pre>
	who 查看登录用户信息	<pre> root@D12B1:~# who am i root pts/0 Mar 9 14:37 (192.168.2.12) root@D12B1:~# whoami root </pre>
	sudo 设置普通用户具有 root 权限	<pre> root@D12B1:~# useradd bigdata root@D12B1:~# passwd bigdata New password: Retype new password: passwd: password updated successfully root@D12B1:~# vi /etc/sudoers root@D12B1:~# su bigdata \$ pwd /root \$ sudo touch 1.txt \$ </pre>
	usermod 修改用户 (modify)	<pre> root@D12B1:~# useradd test root@D12B1:~# usermod -g root test </pre>
	groupadd 新增组	<pre> root@D12B1:~# groupadd testgroup </pre>
	groupdel 删除组	<pre> root@D12B1:~# groupdel testgroup </pre>
	groupmod 修改组	<pre> root@D12B1:~# groupmod -n ttt testgroup </pre>
	cat /etc/group 查看创建了哪些组	<pre> root@D12B1:~# cat /etc/group root:x:0: daemon:x:1: bin:x:2: sys:x:3: adm:x:4: tty:x:5: disk:x:6: lp:x:7: mail:x:8: news:x:9: uucp:x:10: man:x:12: proxy:x:13: kmem:x:15: dialout:x:20: fax:x:21: </pre>

		voice:x:22: cdrom:x:24:vps floppy:x:25:vps tape:x:26: sudo:x:27: audio:x:29:vps dip:x:30:vps www-data:x:33: backup:x:34: operator:x:37: list:x:38: irc:x:39: src:x:40: shadow:x:42: utmp:x:43: video:x:44:vps sasl:x:45: plugdev:x:46:vps staff:x:50: games:x:60: users:x:100:vps nogroup:x:65534: systemd-journal:x:999: systemd-network:x:998: crontab:x:101: input:x:102: sgx:x:103: kvm:x:104: render:x:105: netdev:x:106:vps tss:x:107: systemd-timesync:x:997: messagebus:x:108: _ssh:x:109: ssl-cert:x:110: bluetooth:x:111:vps avahi:x:112: lpadmin:x:113:vps pipewire:x:114: fwupd-refresh:x:115: scanner:x:116:saned,vps saned:x:117: geoclue:x:118: polkitd:x:996: rtkit:x:119: colord:x:120:	
--	--	---	--

		Debian-gdm:x:121: vps:x:1000: gnome-initial-setup:x:995: bigdata:x:1001: test:x:1002: ttt:x:1003:	
文件权限类	文件属性	<pre> root@D12B1:~# ls -l total 4 -rw-r--r-- 1 root root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt </pre>	
	chmod 改变权限	<pre> root@D12B1:~# chmod 777 1.txt root@D12B1:~# ls -l total 4 -rwxrwxrwx 1 root root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt </pre>	
	chown 改变所有者 (属主)	<pre> root@D12B1:~# chown bigdata 1.txt root@D12B1:~# ls -l total 4 -rwxrwxrwx 1 bigdata root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt </pre>	
	chgrp 改变所属组	<pre> root@D12B1:~# chgrp root 1.txt root@D12B1:~# ls -l total 4 -rwxrwxrwx 1 bigdata root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt </pre>	
搜索查找类	find 查找文件或者目录	<pre> root@D12B1:~# find ./ -name 1.txt ./1.txt </pre>	

	locate 快速定位文件路径	<pre> root@D12B1:~# updatedb /usr/bin/find: '/run/user/113/gvfs': Permission denied root@D12B1:~# locate 1.txt /home/vps/.mozilla/firefox/y1a0d2u1.default-esr/pkcs11.txt /root/1.txt /usr/share/doc/debian/constitution.1.1.txt.gz /usr/share/doc/debian/social-contract.1.1.txt.gz /usr/share/doc/kbd/charsets/iso8859-1.txt.gz /usr/share/doc/kbd/charsets/iso8859-11.txt.gz /usr/share/doc/libsane-common/teco/teco1.txt.gz /usr/share/vim/vim90/doc/ft_ps1.txt /usr/share/vim/vim90/doc/gui_x11.txt /usr/share/vim/vim90/doc/usr_01.txt /usr/share/vim/vim90/doc/usr_11.txt /usr/share/vim/vim90/doc/usr_21.txt /usr/share/vim/vim90/doc/usr_31.txt /usr/share/vim/vim90/doc/usr_41.txt /usr/share/vim/vim90/doc/usr_51.txt </pre>	
		<pre> root@D12B1:~# ls grep -n 1.txt 1:1.txt root@D12B1:~# </pre>	
	gzip/gunzip 压缩	<pre> root@D12B1:~# gzip 1.txt root@D12B1:~# ls 1.txt.gz 3.txt 4.txt root@D12B1:~# gunzip 1.txt.gz root@D12B1:~# ls 1.txt 3.txt 4.txt </pre>	
	zip/unzip 压缩	<pre> root@D12B1:~# rm -rf ./* root@D12B1:~# touch 1.txt root@D12B1:~# touch 2.txt root@D12B1:~# zip 3.zip 1.txt 2.txt adding: 1.txt (stored 0%) adding: 2.txt (stored 0%) root@D12B1:~# ls 1.txt 2.txt 3.zip root@D12B1:~# unzip 3.zip Archive: 3.zip replace 1.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: A extracting: 1.txt extracting: 2.txt root@D12B1:~# </pre>	
压缩和解压类			

	tar 打包	<pre> root@D12B1:~# tar -zcvf all.tar.gz ./* ./1.txt ./2.txt ./3.zip root@D12B1:~# ls 1.txt 2.txt 3.zip all.tar.gz root@D12B1:~# rm 1.txt 2.txt 3.zip root@D12B1:~# tar -zxvf all.tar.gz ./1.txt ./2.txt ./3.zip root@D12B1:~# ls 1.txt 2.txt 3.zip all.tar.gz root@D12B1:~# </pre>
系统定时任务	cron 服务管理	<pre> root@D12B1:~# service cron restart </pre>
	crontab 定时任务设置	<pre> root@D12B1:~# crontab -e no crontab for root - using an empty one Select an editor. To change later, run 'select-editor'. 1. /bin/nano <---- easiest 2. /usr/bin/vim.basic 3. /usr/bin/vim.tiny Choose 1-3 [1]: 2 crontab: installing new crontab </pre>
apt 软件包管理	apt update 更新软件包列表	<pre> root@D12B1:~# apt update Hit:1 http://mirrors.ustc.edu.cn/debian bookworm InRelease Hit:2 http://mirrors.ustc.edu.cn/debian bookworm-updates InRelease Hit:3 http://security.debian.org/debian-security bookworm-security InRelease Reading package lists... Done Building dependency tree... Done Reading state information... Done All packages are up to date. </pre>
	apt upgrade 升级软件包	<pre> root@D12B1:~# apt upgrade Reading package lists... Done Building dependency tree... Done Reading state information... Done Calculating upgrade... Done 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded. </pre>

	apt install 安装新的软件包	<pre> root@D12B1:~# apt install nginx Reading package lists... Done Building dependency tree... Done Reading state information... Done The following additional packages will be installed: nginx-common Suggested packages: fcgiwrap nginx-doc The following NEW packages will be installed: nginx nginx-common 0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded. Need to get 640 kB of archives. After this operation, 1696 kB of additional disk space will be used. Do you want to continue? [Y/n] Get:1 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 nginx-common all 1.22.1-9 [112 kB] Get:2 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 nginx amd64 1.22.1-9 [527 kB] Fetched 640 kB in 0s (4555 kB/s) Preconfiguring packages ... Selecting previously unselected package nginx-common. (Reading database ... 157709 files and directories currently installed.) Preparing to unpack .../nginx-common_1.22.1-9_all.deb ... Unpacking nginx-common (1.22.1-9) ... Selecting previously unselected package nginx. Preparing to unpack .../nginx_1.22.1-9_amd64.deb ... Unpacking nginx (1.22.1-9) ... Setting up nginx-common (1.22.1-9) ... Created symlink /etc/systemd/system/multi-user.target.wants/nginx.serv ice -> /lib/systemd/system/nginx.service. Setting up nginx (1.22.1-9) ... Upgrading binary: nginx. Processing triggers for man-db (2.11.2-2) ... root@D12B1:~# </pre>	
	apt remove 卸载软件包	<pre> root@D12B1:~# apt remove nginx Reading package lists... Done Building dependency tree... Done Reading state information... Done The following package was automatically installed and is no longer required: nginx-common Use 'apt autoremove' to remove it. </pre>	

		<p>The following packages will be REMOVED:</p> <p>nginx</p> <p>0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.</p> <p>After this operation, 1362 kB disk space will be freed.</p> <p>Do you want to continue? [Y/n]</p> <p>(Reading database ... 157759 files and directories currently installed.)</p> <p>Removing nginx (1.22.1-9) ...</p> <p>Processing triggers for man-db (2.11.2-2) ...</p> <p>root@D12B1:~#</p>
	apt purge 完全卸载软件包	<p>root@D12B1:~# apt purge nginx</p> <p>Reading package lists... Done</p> <p>Building dependency tree... Done</p> <p>Reading state information... Done</p> <p>Package 'nginx' is not installed, so not removed</p> <p>The following package was automatically installed and is no longer required:</p> <p>nginx-common</p> <p>Use 'apt autoremove' to remove it.</p> <p>0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.</p>
	apt autoremove 自动删除不再需要的软件包	<p>root@D12B1:~# apt autoremove</p> <p>Reading package lists... Done</p> <p>Building dependency tree... Done</p> <p>Reading state information... Done</p> <p>The following packages will be REMOVED:</p> <p>nginx-common</p> <p>0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.</p> <p>After this operation, 334 kB disk space will be freed.</p> <p>Do you want to continue? [Y/n]</p> <p>(Reading database ... 157753 files and directories currently installed.)</p> <p>Removing nginx-common (1.22.1-9) ...</p> <p>root@D12B1:~#</p>
	Shell 编程	<p>helloworld</p> <p>root@D12B1:~# touch helloworld.sh</p> <p>root@D12B1:~# vim helloworld.sh</p> <p>root@D12B1:~# chmod +x helloworld.sh</p> <p>root@D12B1:~# ./helloworld.sh</p> <p>Hello, world!</p>

位置参数变量	<pre>#!/bin/bash greeting="Hello" echo \$greeting echo "Positional Parameters:" echo "1st: \$1" echo "2nd: \$2" echo "3rd: \$3" echo "All positional parameters: \$*" echo "Number of positional parameters: \$#"</pre>
运算符	<pre>root@D12B1:~# S=\$((2+3)*4) root@D12B1:~# echo \$S 20</pre>
判断语句	<pre>root@D12B1:~# ls 1.sh 1.txt 2.txt 3.zip all.tar.gz helloworld.sh root@D12B1:~# [1.txt] && echo true true</pre>
流程控制	<pre>#!/bin/bash if [\$1 -eq "123"] then echo "123" elif [\$1 -eq "456"] then echo "456" else echo "789" fi</pre>
case 语句	<pre>#!/bin/bash case \$1 in "1") echo "1" ;; "2") echo "2" ;; *) echo "other" ;; esac</pre>

	for 循环	<pre>#!/bin/bash for i in "\$*" do echo "The num is \$i " done for j in "\$@" do echo "The num is \$j" done</pre>
	while 循环	<pre>#!/bin/bash s=0 i=1 while [\$i -le 100] do s=\$((s+i)) i=\$((i+1)) done echo \$s</pre>
	read 读取控制台输入	<pre>#!/bin/bash read -t 7 -p "please 7 miao input your name " NAME echo \$NAME</pre>
	系统函数	<pre>root@D12B1:~# basename /root/1.txt 1.txt</pre>
	自定义函数	<pre>#!/bin/bash function sum() { s=0 s=\$((\$1 + \$2)) echo "\$s" } read -p "Please input the number1: " n1; read -p "Please input the number2: " n2; sum \$n1 \$n2;</pre>

(3) Hadoop HDFS 集群环境配置

1 在所有机器安装 JDK8

使用 Adoptium 源安装 temurin-8-jdk

`sudo apt-get install -y wget apt-transport-https gnupg`

`wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | sudo apt-key add -`

`echo "deb https://packages.adoptium.net/artifactory/deb $(lsb_release -sc) main" | sudo`

```
tee /etc/apt/sources.list.d/adoptium.list
sudo apt-get update
sudo apt-get install temurin-8-jdk
```

2 SSH、环境变量、hosts、主机名设置

```
vim /etc/ssh/sshd_config
修改 PermitRootLogin 所在行为 PermitRootLogin yes
```

环境变量

```
echo 'export JAVA_HOME=/usr/lib/jvm/temurin-8-jdk-amd64' >> ~/.bashrc
echo 'export HADOOP_HOME=/usr/local/hadoop' >> ~/.bashrc
echo 'export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin' >> ~/.bashrc
source ~/.bashrc
```

hosts 文件

```
vim /etc/hosts
所有机器文件内容一致。
127.0.0.1 localhost
10.0.3.2 node1
10.0.3.3 node2
10.0.3.4 node3
```

主机名

分别修改 3 台虚拟机的主机名

```
sudo hostnamectl set-hostname node1
sudo hostnamectl set-hostname node2
sudo hostnamectl set-hostname node3
```

修改后 reboot 重启

3 在所有机器下载 Hadoop

```
cd
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
sudo tar zxvf hadoop-3.3.6.tar.gz -C /usr/local/
sudo mv /usr/local/hadoop-3.3.6 /usr/local/hadoop
rm hadoop-3.3.6.tar.gz
```

4 在 node1 配置 SSH 免密登录

在 NameNode 上执行。

编辑 ~/.ssh/config 文件

```
vim ~/.ssh/config
```

填入以下内容:

```
Host node1
```

```
HostName 10.0.3.2
```

Port 22

User root

Host node2

HostName 10.0.3.3

Port 22

User root

Host node3

HostName 10.0.3.4

Port 22

User root

生成密钥并拷贝到 3 台机器上

```
ssh-keygen -t rsa -P ''
```

```
ssh-copy-id node1
```

```
ssh-copy-id node2
```

```
ssh-copy-id node3
```

5 在所有机器配置 Hadoop

hadoop-env.sh

```
vim $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

在文件末尾追加

```
export JAVA_HOME=/usr/lib/jvm/temurin-8-jdk-amd64
```

```
export HDFS_NAMENODE_USER=root
```

```
export HDFS_DATANODE_USER=root
```

```
export HDFS_SECONDARYNAMENODE_USER=root
```

```
export YARN_RESOURCEMANAGER_USER=root
```

```
export YARN_NODEMANAGER_USER=root
```

core-site.xml

```
vim $HADOOP_HOME/etc/hadoop/core-site.xml
```

在 configuration 标签中添加以下内容

```
<!-- 默认文件系统的名称。通过 URI 中 schema 区分不同文件系统 -->
```

```
<!-- file://本地文件系统 hdfs://hadoop 分布式文件系统 -->
```

```
<!-- gfs://google 文件系统 -->
```

```
<!-- hdfs 文件系统访问地址: http://node1:8020 -->
```

```
<property>
```

```
  <name>fs.defaultFS</name>
```

```
  <value>hdfs://node1:8020</value>
```

```
</property>
```

```
<!-- 设置 Hadoop 本地保存数据路径 -->
```



```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/usr/local/hadoop/tmp</value>
</property>

<!-- 设置 HDFS web UI 用户身份 -->
<property>
  <name>hadoop.http.staticuser.user</name>
  <value>root</value>
</property>
```

hdfs-site.xml

vim \$HADOOP_HOME/etc/hadoop/hdfs-site.xml

在 configuration 标签中添加以下内容

```
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
<property>
  <name>dfs.namenode.secondary.http-address</name>
  <value>node1:50090</value>
</property>
```

mapred-site.xml

vim \$HADOOP_HOME/etc/hadoop/mapred-site.xml

在 configuration 标签中添加以下内容

```
<!-- 设置 MR 程序默认运行模式: yarn 集群模式 local 本地模式 -->
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>

<!-- MR 程序历史服务地址 -->
<property>
  <name>mapreduce.jobhistory.address</name>
  <value>node1:10020</value>
</property>

<!-- MR 程序历史服务器 web 端地址 -->
<property>
  <name>mapreduce.jobhistory.webapp.address</name>
  <value>node1:19888</value>
</property>
```

```

<property>
  <name>yarn.app.mapreduce.am.env</name>
  <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>

<property>
  <name>mapreduce.map.env</name>
  <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>

<property>
  <name>mapreduce.reduce.env</name>
  <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>

```

yarn-site.xml

vim \$HADOOP_HOME/etc/hadoop/yarn-site.xml

在 configuration 标签中添加以下内容

```

<!-- 设置 YARN 集群主角色运行机器位置 -->
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>node1</value>
</property>

<!-- ModeManager 上运行的附属服务，需配置成 mapreduce_shuffle 才可运行程序。 -->
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>

```

workers

添加主机名称或 IP

rm \$HADOOP_HOME/etc/hadoop/workers

vim \$HADOOP_HOME/etc/hadoop/workers

node1

node2

node3

7 格式化 NameNode

在 NameNode 上执行格式化命令，只需执行一次

hdfs namenode -format

8 启动 Hadoop 集群

在 NameNode 执行启动命令。如果不成功，可能是 SSH、hosts 文件、主机名配置问题。主机名不能相同。

HDFS 集群

start-dfs.sh

stop-dfs.sh

YARN 集群

start-yarn.sh

stop-yarn.sh

所有集群

start-all.sh

stop-all.sh

9 验证集群状态

在 NameNode 查看 HDFS 集群健康状态

hdfs dfsadmin -report

如果一切顺利，将如下显示：

```
root@node1:~# hdfs dfsadmin -report
```

```
Configured Capacity: 24163061760 (22.50 GB)
```

```
Present Capacity: 5636493312 (5.25 GB)
```

```
DFS Remaining: 5636395008 (5.25 GB)
```

```
DFS Used: 98304 (96 KB)
```

```
DFS Used%: 0.00%
```

```
Replicated Blocks:
```

```
    Under replicated blocks: 0
```

```
    Blocks with corrupt replicas: 0
```

```
    Missing blocks: 0
```

```
    Missing blocks (with replication factor 1): 0
```

```
    Low redundancy blocks with highest priority to recover: 0
```

```
    Pending deletion blocks: 0
```

```
Erasure Coded Block Groups:
```

```
    Low redundancy block groups: 0
```

```
    Block groups with corrupt internal blocks: 0
```

```
    Missing block groups: 0
```

```
    Low redundancy blocks with highest priority to recover: 0
```

```
    Pending deletion blocks: 0
```

```
-----  
Live datanodes (3):
```

```
Name: 10.0.3.2:9866 (node1)
```

```
Hostname: node1
```

```
Decommission Status : Normal
```

```
Configured Capacity: 8054353920 (7.50 GB)
```

```
DFS Used: 32768 (32 KB)
```

Non DFS Used: 5746188288 (5.35 GB)
DFS Remaining: 1876959232 (1.75 GB)
DFS Used%: 0.00%
DFS Remaining%: 23.30%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 0
Last contact: Sun May 26 14:36:19 UTC 2024
Last Block Report: Sun May 26 14:29:56 UTC 2024
Num of Blocks: 0

Name: 10.0.3.3:9866 (node2)
Hostname: node2
Decommission Status : Normal
Configured Capacity: 8054353920 (7.50 GB)
DFS Used: 32768 (32 KB)
Non DFS Used: 5743456256 (5.35 GB)
DFS Remaining: 1879691264 (1.75 GB)
DFS Used%: 0.00%
DFS Remaining%: 23.34%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 0
Last contact: Sun May 26 14:36:17 UTC 2024
Last Block Report: Sun May 26 14:29:49 UTC 2024
Num of Blocks: 0

Name: 10.0.3.4:9866 (node3)
Hostname: node3
Decommission Status : Normal
Configured Capacity: 8054353920 (7.50 GB)
DFS Used: 32768 (32 KB)
Non DFS Used: 5743403008 (5.35 GB)
DFS Remaining: 1879744512 (1.75 GB)
DFS Used%: 0.00%
DFS Remaining%: 23.34%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)

Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 0
Last contact: Sun May 26 14:36:17 UTC 2024
Last Block Report: Sun May 26 14:29:49 UTC 2024
Num of Blocks: 0
也可以前往 HDFS Web UI 界面 <http://node1:9870/>

HomeOverviewDatanodesDataNode Volume FailuresSchedulerStartup ProgressSettings

Overviewnode18020 (active)

Started:Sun May 26 22:28:36 +0800 2024

Version:3.3.6, r16a7828f728a6235a4f98195059f09657247fe

Completed:Sun Jun 16 16:22:00 +0800 2023 by ubuntu from HEAD detached at release-3.3.6-RC3

Cluster ID:CD0-4622750a-4e19-438f-6007-afcc06463814

Block Pool ID:BP-704454638-127.0.1.1-17673779192

Summary

Security is off.
Safemode is off.
1 file and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total Hadoop object(s).
Heap Memory used 36.53 MB of 195 MB Heap Memory. Max Heap Memory is 437.5 MB.
Non Heap Memory used 51.25 MB of 52.0 MB Committed Non-Heap Memory. Max Non-Heap Memory is unbounded.

Configured Capacity:	22.5 GB
Configured Replicate Capacity:	0 B
DFS Used:	96 KB (0%)
Non DFS Used:	16.05 GB
DFS Remaining:	5.25 GB (23.33%)
Block Pool Used:	96 KB (0%)
Datanodes usage% (Min/Median/Max/StdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes:	3 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes:	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes:	0
Entering Maintenance Nodes:	0
Total Datanode Volume Failures:	0 (0 B)
Number of Under-Replicated Blocks:	0
Number of Blocks Pending Deletion (including replicas):	0
Block Deletion Start Time:	Sun May 26 22:28:36 +0800 2024
Last Checkpoint Time:	Sun May 26 21:58:33 +0800 2024
Enabled Erasure Coding Policies:	RS-6-3-1024K

NameNode Journal Status

Current transaction ID: 4
Journal ManagerState
FileJournalManager{root=/usr/local/hadoop/tmp/dfs/name/LOGLogLogOutputStream{/usr/local/hadoop/tmp/dfs/name/current/edits_hpringress_00000000000000000000}}

NameNode Storage

Storage Directory	Type	State
/usr/local/hadoop/tmp/dfs/name	NAME_AND_EDITS	Active

DFS Storage Types

Storage Type	Configured Capacity	Capacity Used	Capacity Remaining	Block Pool Used	Nodes In Service
DISK	22.5 GB	96 KB (0%)	5.25 GB (23.33%)	96 KB	3

Hadoop, 2023.

YARN 集群 UI 界面: <http://node1:8088>

hadoop

Cluster

About

Nodes

Node Labels

Applications

Jobs

Jobs, SAVING

SubMITTED

Running

Running

Failed

Failed

Failed

Failed

Scheduler

Tools

All Applications

Logged in as: root

Cluster Metrics

Apps Submitted	0	Apps Pending	0	Apps Running	0	Apps Completed	0	Containers Running	0	Used Resource	memory:0 B, vCores:0	Total Resource	memory:24 GB, vCores:24	Reserved Resource	memory:0 B, vCores:0	Physical Mem Used %	55	Physical VCores Used %	0
----------------	---	--------------	---	--------------	---	----------------	---	--------------------	---	---------------	----------------------	----------------	-------------------------	-------------------	----------------------	---------------------	----	------------------------	---

Cluster Nodes Metrics

Active Nodes	0	Decommissioning Nodes	0	Decommissioned Nodes	0	Lost Nodes	0	Unhealthy Nodes	0	Rebooted Nodes	0	Shutdown Nodes	0
--------------	---	-----------------------	---	----------------------	---	------------	---	-----------------	---	----------------	---	----------------	---

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority	Scheduler Busy %
Capacity Scheduler	[memory-mb, launch-mb, vcores]	memory:1024, vCores:1	memory:8192, vCores:8	0	0

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	Status	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Stacklisted Nodes
No data available in table																							

Showing 0 to 0 of 0 entries

FirstPreviousNextLast

(4) Hadoop CLI 和 Java API

1 Hadoop CLI

此部分包括 hadoop 常用命令的实操。

常用命令	运行结果
列出目录内容	root@namenode:~# hadoop fs -ls / Found 2 items drwx----- - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user
创建目录	root@namenode:~# hadoop fs -mkdir /directory

删除目录	<pre> root@namenode:~# hadoop fs -rm -r /directory 24/06/01 08:46:47 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes. Deleted /directory </pre>
上传本地文件到HDFS	<pre> root@namenode:~# touch localfile root@namenode:~# hadoop fs -put localfile / root@namenode:~# hadoop fs -ls / Found 3 items -rw-r--r-- 2 root supergroup 0 2024-06-01 08:48 /localfile drwx----- - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user </pre>
从 HDFS 复制文件到本地	<pre> root@namenode:~# rm localfile root@namenode:~# hadoop fs -get / localfile root@namenode:~# ls 1.txt hdfs input localfile run-wordcount.sh start-hadoop.sh </pre>
复制文件	<pre> root@namenode:~# hadoop fs -cp /localfile /copyfile root@namenode:~# hadoop fs -ls / Found 4 items -rw-r--r-- 2 root supergroup 0 2024-06-01 08:51 /copyfile -rw-r--r-- 2 root supergroup 0 2024-06-01 08:48 /localfile drwx----- - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user </pre>
移动文件	<pre> root@namenode:~# hadoop fs -mv /localfile /movefile root@namenode:~# hadoop fs -ls / Found 4 items -rw-r--r-- 2 root supergroup 0 2024-06-01 08:51 /copyfile -rw-r--r-- 2 root supergroup 0 2024-06-01 08:48 /movefile drwx----- - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user </pre>
查看文件	<pre> root@namenode:~# hadoop fs -cat /hello Hello, hadoop! </pre>
更改文件或目录的权限	<pre> root@namenode:~# hadoop fs -chmod 755 /hello </pre>
统计目录中的文件数量和占用空间	<pre> root@namenode:~# hadoop fs -count / 13 10 156456 / </pre>

设置文件的副本数量

```
root@namenode:~# hadoop fs -setrep 5 /hello
Replication 5 set: /hello
```

2 HDFS 的 Java 客户端 API 编程

(1) 在 IDEA 中新建 Java Maven JDK1.8 项目，设置 pom.xml 添加依赖：

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.8.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>2.7.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.7.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-hdfs</artifactId>
    <version>2.7.2</version>
  </dependency>
</dependencies>
```

(2) 创建 HdfsClient 类测试连接

```
public class HdfsClient {
    @Test
    public void check() throws Exception {
//      uploadFile("hello.txt", "/hello.txt");
      printFileList("/");
    }
    public FileSystem getFileSystem() throws Exception {
        Configuration configuration = new Configuration();
        String fileSystemURL = "hdfs://namenode.vayki.com:59000";
        return FileSystem.get(new URI(fileSystemURL), configuration, "root");
    }
}
```

```
}
```

✓ Tests passed: 1 of 1 test – 7 sec 482 ms

/Users/nanmener/Library/Java/JavaVirtualMachines/corretto-1.8.0_382/Contents/Home/bin/java ...

2024-06-03 14:35:28,987 WARN [org.apache.hadoop.util.NativeCodeLoader] - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
directory
hello.txt

Process finished with exit code 0

3 HDFS API 的文件操作

3.1 文件上传

```
public void uploadFile(String source, String destination) throws Exception {  
    FileSystem fs = getFileSystem();  
    fs.copyFromLocalFile(new Path(source), new Path(destination));  
    fs.close();  
}
```

3.2 文件下载

```
public void downloadFile(String hdfsPath, String localPath) throws Exception {  
    FileSystem fs = getFileSystem();  
    fs.copyToLocalFile(new Path(hdfsPath), new Path(localPath));  
    fs.close();  
}
```

3.3 文件夹删除

```
public void deleteDirectory(String directory) throws Exception {  
    FileSystem fs = getFileSystem();  
    fs.delete(new Path(directory), true); // true 表示递归删除  
    fs.close();  
}
```

3.4 文件名更改

```
public void renameFile(String source, String destination) throws Exception {  
    FileSystem fs = getFileSystem();  
    fs.rename(new Path(source), new Path(destination));  
    fs.close();  
}
```

3.5 文件和文件夹判断

```
public void checkFileOrDirectory(String path) throws Exception {  
    FileSystem fs = getFileSystem();  
    FileStatus status = fs.getFileStatus(new Path(path));  
    if (status.isDirectory()) {  
        System.out.println(path + " 是一个文件夹");  
    } else {  
        System.out.println(path + " 是一个文件");  
    }  
    fs.close();  
}
```



```
14 public class HdfsClient {
15     @Test
16     public void check() throws Exception {
17         uploadFile( source: "hello.txt", destination: "/hello.txt");
18         printFileList( directoryPath: "/");
19         downloadFile( hdfsPath: "/hello.txt", localPath: "456.txt");
20         deleteDirectory("/hello.txt");
21         printFileList( directoryPath: "/");
22         renameFile( source: "/directory", destination: "/123");
23         printFileList( directoryPath: "/");
24         checkFileOrDirectory( path: "/");
25         checkFileOrDirectory( path: "/2.txt");
    }
}

>> Tests passed: 1 of 1 test - 2 sec 714 ms
/Users/nanmener/Library/Java/JavaVirtualMachines/corretto-1.8.0_382/Contents/Home/bin/java ...
2024-06-03 14:46:01,998 WARN [org.apache.hadoop.util.NativeCodeLoader] - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
123
2.txt
hello.txt
123
2.txt
123
2.txt
/ 是一个文件夹
/2.txt 是一个文件
```

3.6 I/O 流操作 HDFS

(1) 文件上传

```
public void putFileToHDFS() throws Exception {
    // 1. 获取对象
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf,
    "root");
    // 2. 输入流
    FileInputStream fis = new FileInputStream(new File("testio.txt"));
    // 3. 输出流
    FSDataOutputStream fos = fs.create(new Path("/testio.txt"));
    // 4. 输入输出流相互拷贝
    IOUtils.copyBytes(fis, fos, conf);
    // 5. 关闭流
    IOUtils.closeStream(fos);
    IOUtils.closeStream(fis);
    fs.close();
}
```

(2) 文件下载

```
public void getFileFromHDFS() throws IOException, InterruptedException,
URISyntaxException {
    // 1. 创建配置对象
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf,
    "root");
    // 2. 输入流
    FSDataInputStream fis = fs.open(new Path("/testio.txt"));
    // 3. 输出流
    FileOutputStream fos = new FileOutputStream(new File("testio1.txt"));
    // 4. 流互拷贝
```

```

        IOUtils.copyBytes(fis, fos, conf);
        // 5. 关闭流对象
        IOUtils.closeStream(fos);
        IOUtils.closeStream(fis);
        fs.close();
    }
    (3) 定位文件读取
@Test
/**
 * 下载第 1 块内容
 */
public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException
{
    // 1. 获取对象
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf,
"root");
    // 2. 获取输入流
    FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));
    // 3. 获取输出流
    FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part1"));
    // 4. 流的互拷贝（这里只拷贝指定大小的数据流 128M）
    byte[] buffer = new byte[1024];
    for (int i = 0; i < 1024 * 128; i++) {
        fis.read(buffer);
        fos.write(buffer);
    }
    // 5. 关闭资源
    IOUtils.closeStream(fos);
    IOUtils.closeStream(fis);
    fs.close();
}
/**
 * 下载第 2 块内容
 */
@Test
public void readFileSeek2() throws IOException, InterruptedException, URISyntaxException
{
    // 1. 获取对象
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf,
"root");
    // 2. 获取输入流
    FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));
    // 3. 指定输入流读取位置
    fis.seek(1024*1024*128);

```

```

// 4. 获取输出流
FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part2"));
// 5. 流的互拷贝
IOUtils.copyBytes(fis, fos, conf);
// 6. 关闭资源
IOUtils.closeStream(fos);
IOUtils.closeStream(fis);
fs.close();
}

```

```

68 public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException
69 {
70     // 1. 获取对象
71     Configuration conf = new Configuration();
72     FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf,
73         user: "root");
74     // 2. 获取输入流
75     FSDataInputStream fis = fs.open(new Path(pathString: "/hadoop-3.3.6.tar.gz"));
76     // 3. 获取输出流
77     FileOutputStream fos = new FileOutputStream(new File(pathname: "hadoop-3.3.6.tar.gz
78         .part1"));
79     // 4. 流的互拷贝 (这里只拷贝指定大小的数据流128M)
80     byte[] buffer = new byte[1024];
81     for (int i = 0; i < 1024 * 128; i++) {
82         fis.read(buffer);
83         fos.write(buffer);
84     }
85 }

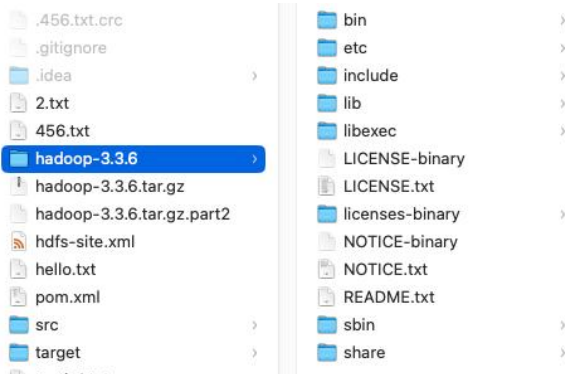
```

测试已通过: 1共 1 个测试 - 14秒 691毫秒
/Users/nanmener/Library/Java/JavaVirtualMachines/corretto-1.8.0_382/Contents/Home/bin/java ...

合并文件

```
cat hadoop-3.3.6.tar.gz.part2 >> hadoop-3.3.6.tar.gz.part1
```

合并完成后进行解压和哈希值比对, 发现与源文件相同。



分析实验方案中“文件写入”的过程

文件写入的流程包括:

- 1) 客户端通过 Distributed FileSystem 模块向 NameNode 请求上传文件, NameNode 检查目标文件是否已存在, 父目录是否存在。
- 2) NameNode 返回是否可以上传。
- 3) 客户端请求第一个 block 上传到哪几个 datanode 服务器上。
- 4) NameNode 返回 2 个 datanode 节点, 分别为 datanode1、datanode2。
- 5) 客户端通过 FSDataOutputStream 模块请求 datanode1 上传数据, datanode1 收到请求会继续调用 datanode2, 将这个通信管道建立完成。
- 6) datanode1、datanode2 逐级应答客户端。
- 7) 客户端开始往 datanode1 上传第一个 block (先从磁盘读取数据放到一个本地内存缓存), 以 packet 为单位, datanode1 收到一个 packet 就会传给 datanode2;

8) 当一个 block 传输完成之后，客户端再次请求 NameNode 上传第二个 block 的服务器。（重复执行 3-7 步）。

(5) NameNode 和 SecondaryNameNode 工作机制

NameNode 和 SecondaryNameNode 是 Hadoop 分布式文件系统中的一个关键组件，它们共同工作以确保文件系统的元数据得到有效管理。

1. NameNode 的工作机制

NameNode 负责管理整个文件系统的元数据，包括文件和目录的命名空间信息以及每个文件的数据块信息。当 NameNode 启动时，如果是首次启动，它会格式化并创建文件系统映像（fsimage）和编辑日志（edits）文件。如果不是首次启动，它会加载 fsimage 和 edits 文件到内存中。客户端对元数据进行增删改的请求时，NameNode 会在内存中对数据进行相应的操作，并记录操作日志到 edits 文件中，但不包括查询操作，因为查询不会更改元数据。

2. SecondaryNameNode 的工作机制

SecondaryNameNode 的主要作用是辅助 NameNode，通过定期合并 fsimage 和 edits 文件来减少 NameNode 的负担。SecondaryNameNode 会询问 NameNode 是否需要执行检查点（checkpoint），这通常是基于时间间隔或 edits 文件大小的阈值来触发的。当触发检查点时，NameNode 会滚动 edits 日志，创建一个新的 edits 文件，并将当前的 edits 和 fsimage 文件发送给 SecondaryNameNode。SecondaryNameNode 接收到这些文件后，会将它们加载到内存中，合并 edits 中的更改到 fsimage 中，生成一个新的 fsimage 文件 fsimage.chkpoint，然后将这个新文件发送回 NameNode。NameNode 接收到新的 fsimage.chkpoint 文件后，会将其重命名为 fsimage，这样在下次启动时就可以使用这个更新后的 fsimage 文件，从而加快启动过程并减少内存中元数据的恢复时间。

SecondaryNameNode 不是 NameNode 的备份，而是一个帮助 NameNode 管理元数据的辅助节点。它通过设置检查点来帮助 NameNode 更有效地工作，确保元数据的一致性和可用性。通过这种方式，NameNode 和 SecondaryNameNode 共同确保了 HDFS 文件系统的稳定性和可靠性。

Fsimage

定义：fsimage 文件系统映像是 HDFS 文件系统的一个快照，它包含了在 NameNode 启动时的文件系统命名空间的状态。这个文件是序列化后的元数据，它记录了所有的文件和目录信息，以及它们的属性和块信息。

作用：fsimage 是 NameNode 内存中元数据的持久化存储形式。在 NameNode 启动时，fsimage 被加载到内存中，以恢复文件系统的命名空间状态。

更新：fsimage 通常在 NameNode 正常启动或通过 SecondaryNameNode 执行检查点操作时更新。

```
root@node1:/usr/local/hadoop/tmp/dfs/name/current# hdfs oiv -i /usr/local/hadoop/tmp/dfs/name/c
urrent/fsimage_0000000000000000005 -o ./fsimage.xml -p XML
2024-06-03 08:17:38,016 INFO offlineImageViewer.FSImageHandler: Loading 2 strings
2024-06-03 08:17:38,496 INFO namenode.FSDirectory: GLOBAL serial map: bits=29 maxEntries=536870
911
2024-06-03 08:17:38,498 INFO namenode.FSDirectory: USER serial map: bits=24 maxEntries=16777215
2024-06-03 08:17:38,498 INFO namenode.FSDirectory: GROUP serial map: bits=24 maxEntries=1677721
5
2024-06-03 08:17:38,498 INFO namenode.FSDirectory: XATTR serial map: bits=24 maxEntries=1677721
5
root@node1:/usr/local/hadoop/tmp/dfs/name/current# vim fsimage.xml
```

Edits

定义：edits 编辑日志记录了自上次 NameNode 启动以来对文件系统的所有修改操作，如文件创建、删除、重命名等。这些操作以事务日志的形式记录，用于追踪对文件系统所做的更改。

作用：edits 的主要目的是记录文件系统状态的变化。在 NameNode 重启时，edits 中的操作会被应用到 fsimage

上，以确保文件系统的状态是最新的。

更新：每当客户端对 HDFS 执行写操作时，相应的操作就会被记录到 edits 文件中。

```
root@node1:/usr/local/hadoop/tmp/dfs/name/current# hdfs oev -i /usr/local/hadoop/tmp/dfs/name/c
urrent/edits_00000000000000000001-00000000000000000002 -o /usr/local/hadoop/tmp/dfs/name/current/
edits_output.xml -p XML
root@node1:/usr/local/hadoop/tmp/dfs/name/current# ls
edits_00000000000000000001-00000000000000000002  fsimage_000000000000000000042
edits_00000000000000000003-00000000000000000003  fsimage_000000000000000000042.md5
edits_00000000000000000004-00000000000000000005  fsimage_0000000000000000000153
edits_00000000000000000006-00000000000000000042  fsimage_0000000000000000000153.md5
edits_000000000000000000043-000000000000000000153  fsimage.xml
edits_inprogress_0000000000000000000154            seen_txid
edits_output.xml                                     VERSION
root@node1:/usr/local/hadoop/tmp/dfs/name/current# vim edits_output.xml
root@node1:/usr/local/hadoop/tmp/dfs/name/current# cat edits_output.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<EDITS>
  <EDITS_VERSION>-66</EDITS_VERSION>
  <RECORD>
    <OPCODE>OP_START_LOG_SEGMENT</OPCODE>
    <DATA>
      <TXID>1</TXID>
    </DATA>
  </RECORD>
  <RECORD>
    <OPCODE>OP_END_LOG_SEGMENT</OPCODE>
    <DATA>
      <TXID>2</TXID>
    </DATA>
  </RECORD>
</EDITS>
```

合并过程 Checkpointing

触发条件：SecondaryNameNode 会根据配置的检查点间隔触发检查点操作。

过程：

- (1)SecondaryNameNode 请求 NameNode 进行检查点。
- (2)NameNode 滚动 edits 文件，创建一个新的 edits 文件（通常是 edits.new），并将当前的 edits 和 fsimage 发送给 SecondaryNameNode。
- (3)SecondaryNameNode 接收到这些文件后，将 edits 中的更改应用到 fsimage 上，生成一个新的 fsimage 文件（通常是 fsimage.chkpoint）。
- (4)SecondaryNameNode 将新的 fsimage 文件发送回 NameNode。
- (5)NameNode 接收新的 fsimage 文件，并将其重命名为当前的 fsimage，同时将 edits.new 重命名为 edits，从而完成检查点操作。

DataNode 工作机制理解

DataNode 是负责存储实际数据的节点。DataNode 通常部署在集群中的多个服务器上，每个 DataNode 负责管理它所在物理服务器的存储。DataNode 的工作机制与 NameNode 密切协作，后者负责管理文件系统的元数据。

当一个文件被上传到 HDFS 时，该文件被切分为多个数据块，然后这些块被分散存储在多个 DataNode 上。每个块通常会有多个副本，分布在不同的 DataNode 上，以提供高可靠性和容错能力。客户端在上传或下载数据时，都是直接与 DataNode 进行交互，而 NameNode 则提供必要的块位置信息。

在正常操作中，DataNode 会定期向 NameNode 发送心跳信号和块报告。心跳信号表明 DataNode 是活跃的，而块报告包含了 DataNode 上所有数据块的详细列表，帮助 NameNode 维护整个文件系统的数据块位置信息。如果 DataNode 失败或其硬盘损坏，NameNode 将依据剩余的副本重新复制数据块到其他 DataNode，以确保数据

的副本数不会低于设定的阈值。

DataNode 参与数据块的校验和计算，以确保数据的完整性。当客户端从 DataNode 读取数据时，DataNode 会计算数据块的校验和，并与存储时计算的校验和进行比对，如果不匹配，说明数据可能已经损坏，系统会尝试从其他 DataNode 获取该数据块的正确副本。

In operation

DataNode State: All

Show 25 entries

Search:

Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Version
✓ /default-rack/node3:9866 (10.0.3.4:9866)	http://node3:9866	0s	136m	376 KB	5.37 GB	7.5 GB	6	376 KB (0%)	3.3.6
✓ /default-rack/node1:9866 (10.0.3.2:9866)	http://node1:9866	1s	86m	341.32 KB	5.35 GB	7.5 GB	6	341.32 KB (0%)	3.3.6
✓ /default-rack/node2:9866 (10.0.3.3:9866)	http://node2:9866	2s	164m	374.99 KB	5.37 GB	7.5 GB	6	374.99 KB (0%)	3.3.6

Showing 1 to 3 of 3 entries

Previous1Next

Hadoop 存档理解

Hadoop 存档是 Hadoop 生态系统中用于高效存储和管理大型数据集的一种机制。它允许用户将 Hadoop 文件系统中的文件或目录打包成一个单一的存档文件，这个文件可以是序列化的形式，从而减少存储空间的使用并提高数据访问速度。存档文件可以包含一个或多个文件，以及目录结构，使得数据的组织和检索更加方便。

当用户创建一个 Hadoop 存档时，系统会将指定的文件或目录压缩并存储在一个单独的文件中。这个过程中，用户可以指定不同的压缩算法，以进一步减少存档文件的大小。Hadoop 存档还支持元数据的存储，比如文件权限、所有者信息和时间戳等，这有助于在恢复数据时保持数据的完整性和一致性。

使用 Hadoop 存档的一个主要优势是它支持并行处理和分布式计算。这意味着存档文件可以被 Hadoop 集群中的多个节点并行读取和处理，从而提高数据处理的效率。存档文件可以被存储在 Hadoop 分布式文件系统上，这为大规模数据集的存储和处理提供了一个稳定和可靠的平台。

在 Hadoop 生态系统中，存档也常用于数据备份和迁移。由于存档文件是自包含的，它们可以轻松地在不同的 Hadoop 集群之间传输，或者作为数据备份存储在不同的存储介质上。存档文件的创建和读取操作都可以通过 Hadoop 命令行工具或编程 API 进行，这为用户提供了灵活性和自动化处理数据的能力。

```
root@node1:~# hadoop archive -archiveName myhar.har -p /directory /
2024-06-03 07:59:54,824 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceM
anager at node1/10.0.3.2:8032
2024-06-03 08:00:01,577 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceM
anager at node1/10.0.3.2:8032
2024-06-03 08:00:01,952 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceM
anager at node1/10.0.3.2:8032
2024-06-03 08:00:03,914 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path:
/tmp/hadoop-yarn/staging/root/.staging/job_1716733951807_0001
2024-06-03 08:00:07,248 INFO mapreduce.JobSubmitter: number of splits:1
2024-06-03 08:00:10,092 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_17167339518
07_0001
2024-06-03 08:00:10,100 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-06-03 08:00:13,553 INFO conf.Configuration: resource-types.xml not found
2024-06-03 08:00:13,560 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-06-03 08:00:17,167 INFO impl.YarnClientImpl: Submitted application application_17167339518
07_0001
2024-06-03 08:00:18,302 INFO mapreduce.Job: The url to track the job: http://node1:8088/proxy/a
pplication_1716733951807_0001/
2024-06-03 08:00:18,434 INFO mapreduce.Job: Running job: job_1716733951807_0001
2024-06-03 08:01:47,502 INFO mapreduce.Job: Job job_1716733951807_0001 running in uber mode : f
alse
2024-06-03 08:01:47,520 INFO mapreduce.Job: map 0% reduce 0%
2024-06-03 08:02:47,545 INFO mapreduce.Job: map 100% reduce 0%
2024-06-03 08:03:30,362 INFO mapreduce.Job: map 100% reduce 100%
2024-06-03 08:03:33,666 INFO mapreduce.Job: Job job_1716733951807_0001 completed successfully
2024-06-03 08:03:34,723 INFO mapreduce.Job: Counters: 54
```

(6) MapReduce 实操

1 WordCount 案例实操

1. 需求

在给定的文本文件中统计输出每一个单词出现的总次数

2. 数据准备

```
root@node1:~# cat input.txt  
aaa bbb ccc bbb bbb aaa
```

3. 编写程序

(1) 编写 mapper 类

```
public class WordcountMapper extends Mapper<LongWritable, Text, Text, IntWritable> { 1个用法  
    Text k = new Text(); 2个用法  
    IntWritable v = new IntWritable( value: 1); 1个用法  
    @Override  
    protected void map(LongWritable key, Text value, Context context)  
        throws IOException, InterruptedException {  
        // 1 获取一行  
        String line = value.toString();  
        // 2 切割  
        String[] words = line.split( regex: " ");  
        // 3 输出  
        for (String word : words) {  
            k.set(word);  
            context.write(k, v);  
        }  
    }  
}
```

(2) 编写 reducer 类

```
public class WordcountReducer extends Reducer<Text, IntWritable, Text, IntWritable> { 1个用法  
    int sum; 3个用法  
    IntWritable v = new IntWritable(); 2个用法  
    @Override  
    protected void reduce(Text key, Iterable<IntWritable> value, Context context) throws IOException, InterruptedException {  
        // 1 累加求和  
        sum = 0;  
        for (IntWritable count : value) {  
            sum += count.get();  
        }  
        // 2 输出  
        v.set(sum);  
        context.write(key, v);  
    }  
}
```

(3) 编写驱动类

```

public class WordcountDriver {
    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {
        // 1 获取配置信息以及封装任务
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        // 2 设置jar加载路径
        job.setJarByClass(WordcountDriver.class);
        // 3 设置map和reduce类
        job.setMapperClass(WordcountMapper.class);
        job.setReducerClass(WordcountReducer.class);
        // 4 设置map输出
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        // 5 设置Reduce输出
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        // 6 设置输入和输出路径
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 7 提交
        job.waitForCompletion(verbose: true);
    }
}

```

4. 集群上测试

(1) 将程序打成 jar 包, 然后拷贝到 hadoop 集群中

```

[INFO] --- compiler:3.11.0:compile (default-compile) @ mapreduce ---
[INFO] Changes detected - recompiling the module! :input tree
[INFO] Compiling 3 source files with javac [debug target 8] to target/classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ mapreduce ---
[INFO] skip non existing resourceDirectory
/Users/nanmener/Projects/mapreduce/src/test/resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ mapreduce ---
[INFO] Changes detected - recompiling the module! :dependency
[INFO]
[INFO] --- surefire:3.2.2:test (default-test) @ mapreduce ---
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ mapreduce ---
[INFO] Building jar: /Users/nanmener/Projects/mapreduce/target/mapreduce-1
.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----

```

(2) 启动 hadoop 集群

(3) 执行 wordcount 程序


```
root@node1:~# ls
input.txt  mapreduce-1.0-SNAPSHOT.jar
root@node1:~# chmod +x mapreduce-1.0-SNAPSHOT.jar
root@node1:~# ls
input.txt  mapreduce-1.0-SNAPSHOT.jar
root@node1:~# hadoop jar mapreduce-1.0-SNAPSHOT.jar com.bigdata.mapreduce.WordcountDriver /input/output1
2024-06-12 05:24:27,975 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at node1/10.0.3.2:8032
2024-06-12 05:24:39,039 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2024-06-12 05:24:39,899 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1716733951807_0002
2024-06-12 05:24:47,306 INFO input.FileInputFormat: Total input files to process : 1
2024-06-12 05:24:49,503 INFO mapreduce.JobSubmitter: number of splits:1
2024-06-12 05:24:55,451 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1716733951807_0002
2024-06-12 05:24:55,452 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-06-12 05:24:59,597 INFO conf.Configuration: resource-types.xml not found
2024-06-12 05:24:59,603 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-06-12 05:25:01,903 INFO impl.YarnClientImpl: Submitted application application_1716733951807_0002
2024-06-12 05:25:03,710 INFO mapreduce.Job: The url to track the job: http://node1:8088/proxy/application_1716733951807_0002/
2024-06-12 05:25:03,729 INFO mapreduce.Job: Running job: job_1716733951807_0002
2024-06-12 05:27:13,295 INFO mapreduce.Job: Job job_1716733951807_0002 running in uber mode : false
2024-06-12 05:27:13,338 INFO mapreduce.Job: map 0% reduce 0%
2024-06-12 05:29:44,701 INFO mapreduce.Job: map 100% reduce 0%
2024-06-12 05:31:21,127 INFO mapreduce.Job: map 100% reduce 67%
2024-06-12 05:31:29,030 INFO mapreduce.Job: map 100% reduce 100%
2024-06-12 05:31:40,831 INFO mapreduce.Job: Job job_1716733951807_0002 completed successfully
```

```
2024-06-12 05:31:43,915 INFO mapreduce.Job: Counters: 54
  File System Counters
    FILE: Number of bytes read=66
    FILE: Number of bytes written=552549
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=122
    HDFS: Number of bytes written=18
    HDFS: Number of read operations=8
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
    HDFS: Number of bytes read erasure-coded=0
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=127614
    Total time spent by all reduces in occupied slots (ms)=104309
    Total time spent by all map tasks (ms)=127614
    Total time spent by all reduce tasks (ms)=104309
    Total vcore-milliseconds taken by all map tasks=127614
    Total vcore-milliseconds taken by all reduce tasks=104309
    Total megabyte-milliseconds taken by all map tasks=130676736
    Total megabyte-milliseconds taken by all reduce tasks=106812416
```

```

Map-Reduce Framework
  Map input records=1
  Map output records=6
  Map output bytes=48
  Map output materialized bytes=66
  Input split bytes=98
  Combine input records=0
  Combine output records=0
  Reduce input groups=3
  Reduce shuffle bytes=66
  Reduce input records=6
  Reduce output records=3
  Spilled Records=12
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=1997
  CPU time spent (ms)=61860
  Physical memory (bytes) snapshot=537894912
  Virtual memory (bytes) snapshot=5175959552
  Total committed heap usage (bytes)=380633088
  Peak Map Physical memory (bytes)=304386048
  Peak Map Virtual memory (bytes)=2585419776
  Peak Reduce Physical memory (bytes)=233508864
  Peak Reduce Virtual memory (bytes)=2590539776

Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=24
File Output Format Counters
  Bytes Written=18

```

```

root@node1:~# hadoop fs -get /output1/part-r-000000 ./
root@node1:~# ls
input.txt  mapreduce-1.0-SNAPSHOT.jar  part-r-000000
root@node1:~# cat part-r-000000
aaa      2
bbb      3
ccc      1
root@node1:~# cat input.txt
aaa bbb ccc bbb bbb aaa
root@node1:~#

```

2 序列化案例实操

1. 需求

统计每一个手机号耗费的总上行流量、下行流量、总流量

2. 编写 mapreduce 程序

(1) 编写流量统计的 bean 对象

```

// 1 实现writable接口
public class FlowBean implements Writable { 0个用法
    private long upFlow ; 6个用法
    private long downFlow; 6个用法
    private long sumFlow; 6个用法
    //2 反序列化时, 需要反射调用空参构造函数, 所以必须有
    public FlowBean() { 0个用法
        super();
    }
    public FlowBean(long upFlow, long downFlow) { 0个用法
        super();
        this.upFlow = upFlow;
        this.downFlow = downFlow;
        this.sumFlow = upFlow + downFlow;
    }
}

```

```

//3 写序列化方法
@Override 0个用法
public void write(DataOutput out) throws IOException {
    out.writeLong(upFlow);
    out.writeLong(downFlow);
    out.writeLong(sumFlow);
}
//4 反序列化方法
//5 反序列化方法读顺序必须和写序列化方法的写顺序必须一致
@Override 0个用法
public void readFields(DataInput in) throws IOException {
    this.upFlow = in.readLong();
    this.downFlow = in.readLong();
    this.sumFlow = in.readLong();
}
// 6 编写toString方法, 方便后续打印到文本
@Override
public String toString() {
    return upFlow + "\t" + downFlow + "\t" + sumFlow;
}
public long getUpFlow() { 0个用法
    return upFlow;
}
public void setUpFlow(long upFlow) { 1个用法
    this.upFlow = upFlow;
}
public long getDownFlow() { 1个用法
    return downFlow;
}

```

(2) 编写 mapper

```

public class FlowCountMapper extends Mapper<LongWritable, Text, Text,
FlowBean>{ 1个用法
    FlowBean v = new FlowBean(); 3个用法
    Text k = new Text(); 2个用法
    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        // 1 获取一行
        String line = value.toString();
        // 2 切割字段
        String[] fields = line.split(regex: "\\t");
        // 3 封装对象
        // 取出手机号码
        String phoneNum = fields[1];
        // 取出上行流量和下行流量
        long upFlow = Long.parseLong(fields[fields.length - 3]);
        long downFlow = Long.parseLong(fields[fields.length - 2]);
        v.setUpFlow(upFlow);
        v.setDownFlow(downFlow);
        k.set(phoneNum);
        // 4 写出
        context.write(k,v);
    }
}

```

(3) 编写 reducer

```

public class FlowCountReducer extends Reducer<Text, FlowBean, Text,
FlowBean> { 0个用法

    @Override
    protected void reduce(Text key, Iterable<FlowBean> values, Context
context)
        throws IOException, InterruptedException {

        long sum_upFlow = 0;
        long sum_downFlow = 0;

        // 1 遍历所用bean, 将其中的上行流量, 下行流量分别累加
        for (FlowBean flowBean : values) {
            sum_upFlow += flowBean.getSumFlow();
            sum_downFlow += flowBean.getDownFlow();
        }

        // 2 封装对象
        FlowBean resultBean = new FlowBean(sum_upFlow, sum_downFlow);

        // 3 写出
        context.write(key, resultBean);
    }
}

```

(4) 编写驱动

```

public class FlowsunDriver {

    public static void main(String[] args) throws IllegalArgumentException,
        IOException, ClassNotFoundException, InterruptedException {
        // 1 获取配置信息, 或者job对象实例
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        // 6 指定本程序的jar包所在的本地路径
        job.setJarByClass(FlowsunDriver.class);
        // 2 指定本业务job要使用的mapper/Reducer业务类
        job.setMapperClass(FlowCountMapper.class);
        job.setReducerClass(FlowCountReducer.class);
        // 3 指定mapper输出数据的kv类型
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(FlowBean.class);
        // 4 指定最终输出的数据的kv类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FlowBean.class);
        // 5 指定job的输入原始文件所在目录
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 7 将job中配置的相关参数, 以及job所用的java类所在的jar包, 提交给yarn去;
        boolean result = job.waitForCompletion(verbose: true);
        System.exit(result ? 0 : 1);
    }
}

```

MapReduce 框架原理

在 MapReduce 的工作流程开始时, 整个数据集被划分为一系列的输入分片, 这些分片被分配到集群中的不同节点上。每个节点运行 Map 函数, Map 函数接收输入分片并将其转换为一系列的中间键值对。这些键值对通常由键和值组成, 键是数据的一个标识符, 而值是与该键相关联的数据。Map 函数的目的是将原始数据转换为一种格式, 使得后续的 Reduce 函数可以更容易地对数据进行汇总。

Map 函数处理完毕后, MapReduce 框架会自动对所有节点生成的中间键值对进行排序和合并。这一步骤是至关重要的, 因为它确保了所有具有相同键的值会被聚集在一起, 为 Reduce 函数的执行提供了必要的前提。排序通常是基于键的字典顺序进行的, 这样可以保证数据的一致性和可预测性。

排序完成后, 数据进入 Reduce 阶段。在这个阶段, 每个唯一的键及其对应的所有值被传递给 Reduce 函数。Reduce 函数的目的是将这些值进行汇总或合并, 以生成最终的输出。例如, 如果 Map 函数输出的是每个单词出现的次数, Reduce 函数可能会计算所有单词的总数, 或者找出出现次数最多的单词。

MapTask 工作机制

MapTask 从输入分片开始。输入数据通常存储在分布式文件系统中, 系统会将大文件分割成多个分片, 每个分片由一个 MapTask 来处理。每个输入分片会被分配给一个 MapTask, MapTask 会从分片中读取数据。

MapTask 读取数据时, 会先将数据解析成记录, 这些记录通常是键值对。在处理文本文件时, 每一行文本可以被视为一个记录。输入格式决定了数据如何被解析成键值对。常见的输入格式包括 TextInputFormat、KeyValueTextInputFormat 等。

MapTask 对读取到的记录应用用户定义的 Map 函数。Map 函数是用户在 MapReduce 程序中编写的逻辑，用于处理每一个输入键值对，并生成零个或多个中间键值对。这些中间键值对是 Map 阶段的输出，供后续的 Shuffle 和 Sort 阶段使用。

在 Map 函数处理完所有输入记录后，MapTask 会将生成的中间键值对临时存储在内存中。当内存中的数据达到一定阈值时，MapTask 会将数据溢写到磁盘上。溢写过程包括对中间键值对进行排序和分区，以便后续的 Shuffle 和 Sort 阶段能高效进行。

MapTask 会创建多个分区，每个分区对应一个 ReduceTask。MapTask 在将中间键值对写入磁盘时，会按照键值对的键进行分区，这样相同键的键值对会被发送到同一个分区。分区的数量通常等于 ReduceTask 的数量，用户可以通过 Partitioner 函数来自定义分区逻辑。

在所有输入数据都被处理完，并且所有中间键值对都被写入磁盘后，MapTask 会将这些分区文件传输给相应的 ReduceTask。这一过程称为 Shuffle 阶段，MapTask 会将中间数据传输到 Reduce 节点上。

Shuffle 机制

在 Map 阶段完成后，每个 Map 任务会产生一组中间键值对。首先，Map 任务会将这些键值对根据键的哈希值进行分区，不同的键被分配到不同的分区，每个分区对应一个 Reduce 任务。这样做的目的是确保相同的键最终会被同一个 Reduce 任务处理。用户可以自定义分区逻辑，通过实现 Partitioner 接口来控制数据分区方式。

Map 任务会对每个分区中的键值对进行排序。排序的目的是将相同的键聚集在一起，便于 Reduce 任务后续处理。这一步骤通常在内存中进行，当内存中存储的数据达到阈值时，会触发溢写操作，将数据写入磁盘。溢写时的数据是有序的，多个溢写文件会被合并成一个大的有序文件。

所有的 Map 任务完成并产生中间数据，Shuffle 阶段便正式开始。每个 Reduce 任务会从所有的 Map 任务中获取它所需的分区数据。这涉及到网络传输，因为 Map 任务和 Reduce 任务通常在不同的节点上运行。为了减小网络带宽的消耗，Shuffle 机制会对数据进行压缩。Map 任务在发送数据之前会先压缩数据，Reduce 任务在接收数据后会解压缩。

在 Reduce 节点，Shuffle 机制会从不同的 Map 任务接收到相同分区的数据，这些数据仍然是有序的。Reduce 任务会将这些数据进行合并，确保所有键值对按照键排序并准备好进行 Reduce 阶段的处理。这一过程需要处理大量的网络传输和磁盘 I/O 操作，因此 Shuffle 机制的效率直接影响到整个 MapReduce 作业的性能。

Reduce join 案例实操

- 1) 创建商品和订单合并后的 bean 类

```

public class TableBean implements Writable { 15 个用法
    private String order_id; // 订单id 6 个用法
    private String p_id; // 产品id 5 个用法
    private int amount; // 产品数量 6 个用法
    private String pname; // 产品名称 6 个用法
    private String flag; // 表的标记 5 个用法

    public TableBean() { 3 个用法
        super();
    }

    public TableBean(String order_id, String p_id, int amount, String
    pname, String flag) { 0 个用法
        super();
        this.order_id = order_id;
        this.p_id = p_id;
        this.amount = amount;
        this.pname = pname;
        this.flag = flag;
    }

    public String getFlag() { 1 个用法
        return flag;
    }

    public void setFlag(String flag) { 2 个用法
        this.flag = flag;
    }

    public String getOrder_id() { 0 个用法
        return order_id;
    }

    public void setOrder_id(String order_id) { 2 个用法
        this.order_id = order_id;
    }

    public String getP_id() { 0 个用法
        return p_id;
    }

    public void setP_id(String p_id) { 2 个用法
        this.p_id = p_id;
    }

    public int getAmount() { 0 个用法
        return amount;
    }

    public void setAmount(int amount) { 2 个用法
        this.amount = amount;
    }

    public String getPname() { 1 个用法
        return pname;
    }
}

```

```

@Override
public void write(DataOutput out) throws IOException {
    out.writeUTF(order_id);
    out.writeUTF(p_id);
    out.writeInt(amount);
    out.writeUTF(pname);
    out.writeUTF(flag);
}

@Override
public void readFields(DataInput in) throws IOException {
    this.order_id = in.readUTF();
    this.p_id = in.readUTF();
    this.amount = in.readInt();
    this.pname = in.readUTF();
    this.flag = in.readUTF();
}

@Override
public String toString() {
    return order_id + "\t" + pname + "\t" + amount + "\t" ;
}

```

2) 编写 TableMapper 程序

```

public class TableMapper extends Mapper<LongWritable, Text, Text, TableBean>{ 1个用法
    TableBean bean = new TableBean(); 11个用法
    Text k = new Text(); 3个用法

    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        // 1 获取输入文件类型
        FileSplit split = (FileSplit) context.getInputSplit();
        String name = split.getPath().getName();
        // 2 获取输入数据
        String line = value.toString();
        // 3 不同文件分别处理
        if (name.startsWith("order")) { // 订单表处理
            // 3.1 切割
            String[] fields = line.split(regex: "\t");
            // 3.2 封装bean对象
            bean.setOrder_id(fields[0]);
            bean.setP_id(fields[1]);
            bean.setAmount(Integer.parseInt(fields[2]));
            bean.setPname("");
            bean.setFlag("0");
            k.set(fields[1]);
        } else { // 产品表处理

```



```

    }else { // 产品表处理
        // 3.3 切割
        String[] fields = line.split( regex: "\\t");
        // 3.4 封装bean对象
        bean.setP_id(fields[0]);
        bean.setName(fields[1]);
        bean.setFlag("1");
        bean.setAmount(0);
        bean.setOrder_id("");
        k.set(fields[0]);
    }
    // 4 写出
    context.write(k, bean);
}
}

```

3) 编写 TableReducer 程序

```

public class TableReducer extends Reducer<Text, TableBean, TableBean, NullWritable>
{
    1 个用法
    @Override
    protected void reduce(Text key, Iterable<TableBean> values, Context context)
        throws IOException, InterruptedException {
        // 1 准备存储订单的集合
        ArrayList<TableBean> orderBeans = new ArrayList<>();
        // 2 准备bean对象
        TableBean pdBean = new TableBean();
        for (TableBean bean : values) {
            if ("0".equals(bean.getFlag())) { // 订单表
                // 拷贝传递过来的每条订单数据到集合中
                TableBean orderBean = new TableBean();
                try {
                    BeanUtils.copyProperties(orderBean, bean);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                orderBeans.add(orderBean);
            } else { // 产品表
                try {
                    // 拷贝传递过来的产品表到内存中
                    BeanUtils.copyProperties(pdBean, bean);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        // 3 表的拼接
        for (TableBean bean:orderBeans){
            bean.setName (pdBean.getName());
            // 4 数据写出去
            context.write(bean, NullWritable.get());
        }
    }
}
}

```

4) 编写 TableDriver 程序

```

public class TableDriver {

    public static void main(String[] args) throws Exception {
        // 1 获取配置信息, 或者job对象实例
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        // 2 指定本程序的jar包所在的本地路径
        job.setJarByClass(TableDriver.class);
        // 3 指定本业务job要使用的mapper/Reducer业务类
        job.setMapperClass(TableMapper.class);
        job.setReducerClass(TableReducer.class);
        // 4 指定mapper输出数据的kv类型
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(TableBean.class);
        // 5 指定最终输出的数据的kv类型
        job.setOutputKeyClass(TableBean.class);
        job.setOutputValueClass(NullWritable.class);
        // 6 指定job的输入原始文件所在目录
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 7 将job中配置的相关参数, 以及job所用的java类所在的jar包, 提交给yarn去运行
        boolean result = job.waitForCompletion(verbose: true);
        System.exit(result ? 0 : 1);
    }
}

```

5) 运行程序查看结果

```

[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ table ---
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ table ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 4 source files with javac [debug target 8] to target/classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ table ---
[INFO] skip non existing resourceDirectory
/Users/nanmener/Projects/table/src/test/resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ table ---
[INFO] Changes detected - recompiling the module! :dependency
[INFO]
[INFO] --- surefire:3.2.2:test (default-test) @ table ---
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ table ---
[INFO] Building jar: /Users/nanmener/Projects/table/target/table-1.0-SNAPSHOT
.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----

```

```

root@node1:~/table_input# hadoop jar table_join.jar com.bigdata.mapreduce.table.TableDriver /table_input /table_output

2024-06-12 15:11:57,590 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using built-in
es where applicable
2024-06-12 15:11:57,978 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at node1/10.0.4.11
2024-06-12 15:11:58,213 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement
face and execute your application with ToolRunner to remedy this.
2024-06-12 15:11:58,228 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
job_1718175323880_0002
2024-06-12 15:11:58,455 INFO input.FileInputFormat: Total input files to process : 2
2024-06-12 15:11:58,525 INFO mapreduce.JobSubmitter: number of splits:2
2024-06-12 15:11:58,638 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1718175323880_0002
2024-06-12 15:11:58,639 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-06-12 15:11:58,761 INFO conf.Configuration: resource-types.xml not found
2024-06-12 15:11:58,761 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-06-12 15:11:58,822 INFO impl.YarnClientImpl: Submitted application application_1718175323880_0002
2024-06-12 15:11:58,854 INFO mapreduce.Job: The url to track the job: http://node1:8088/proxy/application_1718175323880_
2024-06-12 15:11:58,855 INFO mapreduce.Job: Running job: job_1718175323880_0002
2024-06-12 15:12:02,934 INFO mapreduce.Job: Job job_1718175323880_0002 running in uber mode : false
2024-06-12 15:12:02,935 INFO mapreduce.Job: map 0% reduce 0%
2024-06-12 15:12:07,040 INFO mapreduce.Job: map 100% reduce 0%
2024-06-12 15:12:12,142 INFO mapreduce.Job: map 100% reduce 100%
2024-06-12 15:12:12,161 INFO mapreduce.Job: Job job_1718175323880_0002 completed successfully

2024-06-12 15:12:12,236 INFO mapreduce.Job: Counters: 54
  File System Counters
    FILE: Number of bytes read=156
    FILE: Number of bytes written=829037
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=270
    HDFS: Number of bytes written=45
    HDFS: Number of read operations=11
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
    HDFS: Number of bytes read erasure-coded=0
  Job Counters
    Launched map tasks=2
    Launched reduce tasks=1
    Data-local map tasks=2
    Total time spent by all maps in occupied slots (ms)=3892
    Total time spent by all reduces in occupied slots (ms)=1503
    Total time spent by all map tasks (ms)=3892
    Total time spent by all reduce tasks (ms)=1503
    Total vcore-milliseconds taken by all map tasks=3892
    Total vcore-milliseconds taken by all reduce tasks=1503
    Total megabyte-milliseconds taken by all map tasks=3985408
    Total megabyte-milliseconds taken by all reduce tasks=1539072
  Map-Reduce Framework
    Map input records=6
    Map output records=6
    Map output bytes=138
    Map output materialized bytes=162
    Input split bytes=210
    Combine input records=0
    Combine output records=0
    Reduce input groups=3
    Reduce shuffle bytes=162
    Reduce input records=6
    Reduce output records=3
    Spilled Records=12
    Reduce output records=3
    Spilled Records=12
    Shuffled Maps =2
    Failed Shuffles=0
    Merged Map outputs=2
    GC time elapsed (ms)=124
    CPU time spent (ms)=1170
    Physical memory (bytes) snapshot=831242240
    Virtual memory (bytes) snapshot=7429001216
    Total committed heap usage (bytes)=681050112
    Peak Map Physical memory (bytes)=304803840
    Peak Map Virtual memory (bytes)=2473291776
    Peak Reduce Physical memory (bytes)=224030720
    Peak Reduce Virtual memory (bytes)=2482688000
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=60
  File Output Format Counters
    Bytes Written=45
root@node1:~/table_input# hdfs dfs -cat /output/part-r-00000
2024-06-12 15:12:31,623 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using built
es where applicable
cat: '/output/part-r-00000': No such file or directory
root@node1:~/table_input# hdfs dfs -cat /table_output/part-r-00000
2024-06-12 15:12:39,707 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using built
es where applicable
1001 小米 1
1002 华为 2
1003 格力 3

```

(7) Yarn

Yarn 基本架构

Yarn 是 Hadoop 生态系统中的一个关键组件，负责集群资源的管理和作业调度。它通过将资源管理与作业调度分离，解决了原有 Hadoop MapReduce 的扩展性问题。Yarn 的基本架构由以下几个主要组件构成：资源管理器、节点管理器、应用程序主控和容器。

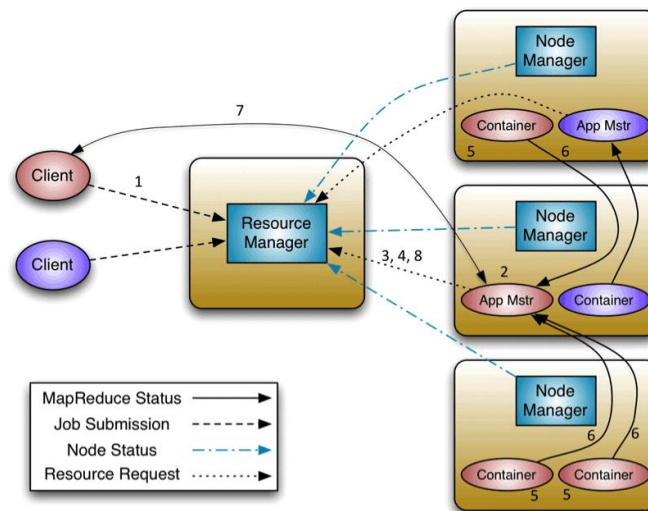
资源管理器 Resource Manager 是整个集群的中央控制组件，负责接收作业请求、分配资源和调度任务。它包括两个关键部分：调度器 Scheduler 和应用管理器 Application Manager。调度器根据可用资源和预定义的策略，动态地分配资源给各个应用程序，而不直接涉及任务的具体执行。应用管理器则负责管理应用程序的生命周期，包括启动应用程序主控、监控应用程序运行状态等。

节点管理器 Node Manager 运行在集群中的每个节点上，负责管理该节点上的资源使用情况和任务执行。它定期向资源管理器汇报节点的资源使用情况，并接收资源管理器的指令来启动或停止容器。节点管理器负责监控容器的运行状态和资源使用情况，确保任务在规定的资源限制内运行。

应用程序主控 Application Master 是每个应用程序独有的组件，负责整个应用程序的任务调度和监控。应用程序主控在启动时向资源管理器请求资源，并在获得资源后，在相应的节点上启动任务容器。它负责处理任务失败的重试、任务进度的跟踪等，确保应用程序的正常执行。

容器 Container 是 Yarn 中资源分配的基本单元，包含特定数量的 CPU、内存和其他资源。应用程序主控从资源管理器请求到的资源以容器的形式分配，节点管理器负责在具体的节点上启动这些容器，并将任务分配给它们执行。每个容器运行一个任务进程，并在任务完成后释放资源。

Yarn 工作机制



Yarn 的工作机制涉及多个步骤和组件之间的交互，确保作业能够高效地在集群中执行。首先，用户提交作业时，作业描述和资源需求被发送到资源管理器。资源管理器作为集群的中央控制组件，接收到作业请求后，会将资源请求分配给相应的节点管理器，并启动应用程序主控 Application Master。

应用程序主控是每个作业独有的组件，它在获得资源后启动，并负责管理作业整个生命周期。应用程序主控首先在资源管理器上注册，并向资源管理器请求必要的资源来执行作业。资源管理器通过调度器来分配这些资源，调度器根据当前集群资源的使用情况和预定义的策略来决定如何分配资源。

应用程序主控获得容器后，将任务分配给这些容器执行。每个任务在一个容器内运行，应用程序主控负责监控任务的执行状态，并处理任务失败的重试和任务进度跟踪。如果任务失败，应用程序主控会根据策略决定是否重试任务或者请求更多资源。

作业完成后，应用程序主控会向资源管理器注销，并释放所有占用的资源，节点管理器停止并清理相关的容器，确保资源能够被其他作业使用。资源管理器更新作业的状态，并将结果反馈给用户，整个作业流程结束。

资源调度器

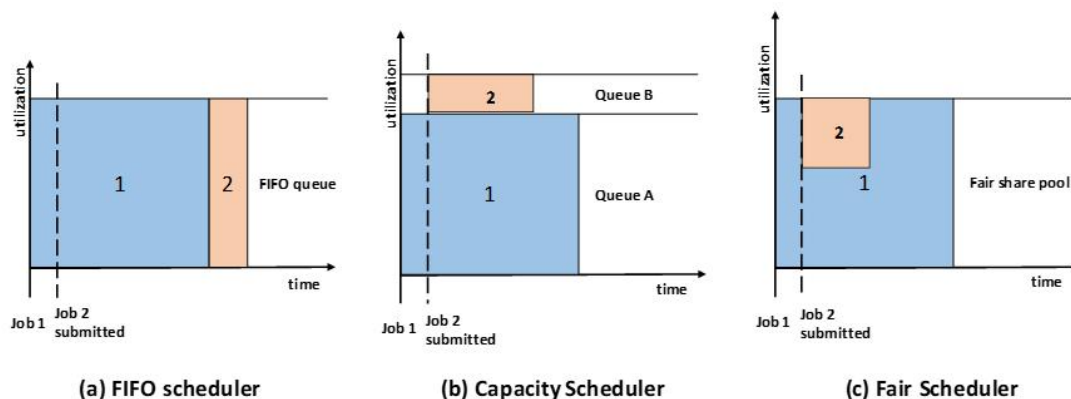


Figure 1: YARN Schedulers' cluster utilization vs. time

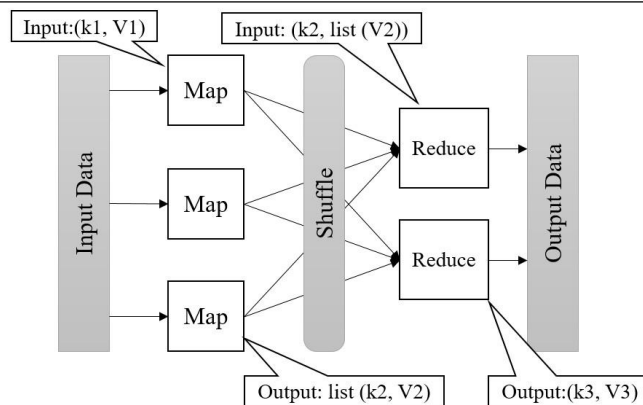
YARN 中的资源调度器是资源管理器 Resource Manager 的一个关键组件，负责决定如何将集群中的可用资源分配给多个应用程序和作业。资源调度器不直接执行任务，而是根据集群资源的状态和调度策略，动态分配资源。YARN 默认提供了几种调度器，包括 FIFO 调度器、容量调度器 Capacity Scheduler 和公平调度器 Fair Scheduler，每种调度器都有不同的特点和适用场景。

FIFO 调度器 First In First Out Scheduler 是最简单的一种调度器，它按照作业提交的先后顺序分配资源。FIFO 调度器的优点是实现简单，但它不考虑作业的资源需求和执行时间，可能导致某些大作业长时间占用资源，从而延迟其他作业的执行。

容量调度器 Capacity Scheduler 旨在支持多租户环境，确保资源在不同的队列之间按配置的容量比例分配。每个队列可以配置不同的容量和优先级，队列内部的作业按照 FIFO 顺序调度。容量调度器允许用户根据业务需求灵活配置资源使用策略，并提供资源保障机制，确保关键作业能够获得足够的资源。它支持队列的层级结构，可以对队列进行子队列划分，以更细粒度地管理资源。

公平调度器 Fair Scheduler 通过动态调整资源分配，使每个作业都能获得公平的资源份额。它会平衡资源使用，防止某些作业长时间占用资源而导致其他作业得不到足够资源。公平调度器支持多种调度策略，例如基于资源份额的分配和基于作业优先级的分配，还可以配置资源池，确保不同类型的作业得到合理的资源分配。它适合在资源竞争激烈的环境中使用，能够提高集群资源的整体利用率和作业的响应速度。

MapReduce 优化方法



优化数据的布局 and 分区。优化输入数据的格式和分区方式，可以减少数据传输和 I/O 操作。将数据预先分区，确保数据块大小均衡，避免数据倾斜问题。使用合适的文件格式可以提高读取和写入性能。

合理配置 MapReduce 作业的参数。作业的分片数量 splits 和每个分片的大小 split size 直接影响任务的并行度和执行时间。设置合理的 map 和 reduce 任务数量，避免任务过多导致调度开销增大或任务过少导致资源浪费。调整任务的内存和 CPU 资源配置，如 `mapreduce.map.memory.mb` 和 `mapreduce.reduce.memory.mb` 参数，确保任务在资源充足的情况下运行。

优化 Mapper 和 Reducer 的实现。在 Mapper 中尽量减少复杂的计算逻辑和外部资源的访问，确保快速处理输入数据。使用 Combiner 在 Mapper 阶段进行部分聚合，减少传输到 Reducer 的数据量。Reducer 的实现要注意优化聚合逻辑，避免使用全局排序等高开销操作。

数据的本地化处理。尽量将计算任务调度到数据所在的节点，减少数据传输的开销。配置集群的调度策略，使其优先考虑数据本地化，将计算任务分配到存储数据的节点上。

(8) MapReduce 综合

需求：对每一个 maptask 的输出局部汇总

统计过程中对每一个 maptask 的输出进行局部汇总，以减小网络传输量即采用 Combiner 功能。

增加一个 WordcountCombiner 类继承 Reducer

```

public class WordcountReducer extends Reducer<Text, IntWritable,
Text, IntWritable>{ 1个用法
    int sum; 3个用法
    IntWritable v = new IntWritable(); 2个用法
    @Override
    protected void reduce(Text key, Iterable<IntWritable> value,
        Context context) throws IOException,
        InterruptedException {

        // 1 累加求和
        sum = 0;
        for (IntWritable count : value) {
            sum += count.get();
        }
        // 2 输出
        v.set(sum);
        context.write(key,v);
    }
}
  
```

```

Map-Reduce Framework
  Map input records=7
  Map output records=14
  Map output bytes=146
  Map output materialized bytes=162
  Input split bytes=296
  Combine input records=14
  Combine output records=11
  Reduce input groups=11
  Reduce shuffle bytes=162
  Reduce input records=11
  Reduce output records=11
  Spilled Records=22
  Shuffled Maps =3
  Failed Shuffles=0
  Merged Map outputs=3
  GC time elapsed (ms)=216
  CPU time spent (ms)=1660
  Physical memory (bytes) snapshot=1147633664
  Virtual memory (bytes) snapshot=9907957760
  Total committed heap usage (bytes)=917504000
  Peak Map Physical memory (bytes)=308846592
  Peak Map Virtual memory (bytes)=2474262528
  Peak Reduce Physical memory (bytes)=225751040
  Peak Reduce Virtual memory (bytes)=2487021568

```

需求 4: 大量小文件的切片优化

将输入的大量小文件合并成一个切片统一处理。

在 WordcountDriver 中增加如下代码

```

// 如果不设置InputFormat, 它默认用的是TextInputFormat.class
job.setInputFormatClass(CombineTextInputFormat.class);
CombineTextInputFormat.setMaxInputSplitSize(job,
    size: 4194304); // 4m
CombineTextInputFormat.setMinInputSplitSize(job,
    size: 2097152); // 2m

```

运行程序, 并观察运行的切片个数为 1

```

input.FileInputFormat: Total input files to process : 3
mapreduce.JobSubmitter: number of splits:1
mapreduce.JobSubmitter: Submitting tokens for job: job_1718175323880_0005
mapreduce.JobSubmitter: Executing with tokens: []

```

流量汇总案例

需求 2: 将统计结果按照手机归属地不同省份输出到不同文件中

(1) Mapreduce 中会将 map 输出的 kv 对, 按照相同 key 分组, 然后分发给不同的 reducer task。默认的分发规则为: 根据 key 的 hashCode%reducer task 数来分发

(2) 如果要按照我们自己的需求进行分组, 则需要改写数据分发 (分组) 组件 Partitioner

自定义一个 CustomPartitioner 继承抽象类: Partitioner

(3) 在 job 驱动中, 设置自定义 partitioner: job.setPartitionerClass(CustomPartitioner.class)

在需求 1 的基础上, 增加一个分区类

```

public class ProvincePartitioner extends Partitioner<Text, FlowBean> { 0 个用法

    @Override 0 个用法
    public int getPartition(Text key, FlowBean value, int numPartitions) {
        // 1 获取电话号码的前三位
        String preNum = key.toString().substring(0, 3);

        int partition = 4;

        // 2 判断是哪个省
        if ("136".equals(preNum)) {
            partition = 0;
        } else if ("137".equals(preNum)) {
            partition = 1;
        } else if ("138".equals(preNum)) {
            partition = 2;
        } else if ("139".equals(preNum)) {
            partition = 3;
        }

        return partition;
    }
}

```

在驱动函数中增加自定义数据分区设置和 reduce task 设置

```

public static void main(String[] args) throws IllegalArgumentException, IOException,
    ClassNotFoundException, InterruptedException {
    // 1 获取配置信息，或者job对象实例
    Configuration configuration = new Configuration();
    Job job = Job.getInstance(configuration);
    // 6 指定本程序的jar包所在的本地路径
    job.setJarByClass(FlowsumDriver.class);
    // 2 指定本业务job要使用的mapper/Reducer业务类
    job.setMapperClass(FlowCountMapper.class);
    job.setReducerClass(FlowCountReducer.class);
    // 3 指定mapper输出数据的kv类型
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(FlowBean.class);
    // 4 指定最终输出的数据的kv类型
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(FlowBean.class);
    // 8 指定自定义数据分区
    job.setPartitionerClass(ProvincePartitioner.class);
    // 9 同时指定相应数量的reduce task
    job.setNumReduceTasks(5);
    // 5 指定job的输入原始文件所在目录
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    // 7 将job中配置的相关参数，以及job所用的java类所在的jar包， 提交给yarn去运行
    boolean result = job.waitForCompletion(verbose: true);
    System.exit(result ? 0 : 1);
}

```


File System Counters

FILE: Number of bytes read=68
FILE: Number of bytes written=1658861
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=198
HDFS: Number of bytes written=22
HDFS: Number of read operations=28
HDFS: Number of large read operations=0
HDFS: Number of write operations=10
HDFS: Number of bytes read erasure-coded=0

Job Counters

Launched map tasks=1
Launched reduce tasks=5
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=1982
Total time spent by all reduces in occupied slots (ms)=19906
Total time spent by all map tasks (ms)=1982
Total time spent by all reduce tasks (ms)=19906
Total vcore-milliseconds taken by all map tasks=1982
Total vcore-milliseconds taken by all reduce tasks=19906
Total megabyte-milliseconds taken by all map tasks=2029568
Total megabyte-milliseconds taken by all reduce tasks=20383744

Map-Reduce Framework

Map input records=1
Map output records=1
Map output bytes=36
Map output materialized bytes=68
Input split bytes=111
Combine input records=0
Combine output records=0
Reduce input groups=1
Reduce shuffle bytes=68
Reduce input records=1
Reduce output records=1
Spilled Records=2
Shuffled Maps =5
Failed Shuffles=0
Merged Map outputs=5
GC time elapsed (ms)=428
CPU time spent (ms)=2730
Physical memory (bytes) snapshot=1346945024
Virtual memory (bytes) snapshot=14884687872
Total committed heap usage (bytes)=1015545856
Peak Map Physical memory (bytes)=307638272
Peak Map Virtual memory (bytes)=2473943040
Peak Reduce Physical memory (bytes)=223698944
Peak Reduce Virtual memory (bytes)=2484613120

Shuffle Errors





BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters

Bytes Read=87

File Output Format Counters

Bytes Written=22

Replication	Block Size	Name	
3	128 MB	_SUCCESS	
3	128 MB	part-r-00000	
3	128 MB	part-r-00001	
3	128 MB	part-r-00002	
3	128 MB	part-r-00003	
3	128 MB	part-r-00004	

需求 3: 将统计结果按照总流量倒序排序 (全排序)

- (1) 把程序分两步走, 第一步正常统计总流量, 第二步再把结果进行排序
- (2) context.write(总流量, 手机号)
- (3) FlowBean 实现 WritableComparable 接口重写 compareTo 方法

(1) FlowBean 对象在需求 1 基础上增加了比较功能

```
package com.bigdata.mapreduce.sort;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.WritableComparable;

public class FlowBean implements WritableComparable<FlowBean> {

    private long upFlow;
    private long downFlow;
    private long sumFlow;

    // 反序列化时, 需要反射调用空参构造函数, 所以必须有
    public FlowBean() {
        super();
    }

    public FlowBean(long upFlow, long downFlow) {
        super();
        this.upFlow = upFlow;
        this.downFlow = downFlow;
        this.sumFlow = upFlow + downFlow;
    }

    public void set(long upFlow, long downFlow) {
        this.upFlow = upFlow;
        this.downFlow = downFlow;
        this.sumFlow = upFlow + downFlow;
    }
}
```

```
public long getSumFlow() {
    return sumFlow;
}

public void setSumFlow(long sumFlow) {
    this.sumFlow = sumFlow;
}

public long getUpFlow() {
    return upFlow;
}

public void setUpFlow(long upFlow) {
    this.upFlow = upFlow;
}

public long getDownFlow() {
    return downFlow;
}

public void setDownFlow(long downFlow) {
    this.downFlow = downFlow;
}

/**
 * 序列化方法
 * @param out
 * @throws IOException
 */
@Override
public void write(DataOutput out) throws IOException {
    out.writeLong(upFlow);
    out.writeLong(downFlow);
    out.writeLong(sumFlow);
}

/**
 * 反序列化方法 注意反序列化的顺序和序列化的顺序完全一致
 * @param in
 * @throws IOException
 */
@Override
public void readFields(DataInput in) throws IOException {
    upFlow = in.readLong();
    downFlow = in.readLong();
    sumFlow = in.readLong();
}
```

```

    }

    @Override
    public String toString() {
        return upFlow + "\t" + downFlow + "\t" + sumFlow;
    }

    @Override
    public int compareTo(FlowBean o) {
        // 倒序排列, 从大到小
        return this.sumFlow > o.getSumFlow() ? -1 : 1;
    }
}

```

(2) 编写 mapper

```

package com.bigdata.mapreduce.sort;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class FlowCountSortMapper extends Mapper<LongWritable, Text, FlowBean, Text>{
    FlowBean bean = new FlowBean();
    Text v = new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        // 1 获取一行
        String line = value.toString();

        // 2 截取
        String[] fields = line.split("\\s+");

        // 3 封装对象
        String phoneNbr = fields[0];
        long upFlow = Long.parseLong(fields[1]);
        long downFlow = Long.parseLong(fields[2]);

        bean.set(upFlow, downFlow);
        v.set(phoneNbr);

        // 4 输出
        context.write(bean, v);
    }
}

```

(3) 编写 reducer

```
package com.bigdata.mapreduce.sort;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class FlowCountSortReducer extends Reducer<FlowBean, Text, Text, FlowBean>{

    @Override
    protected void reduce(FlowBean key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {

        // 循环输出，避免总流量相同情况
        for (Text text : values) {
            context.write(text, key);
        }
    }
}
```

(4) 编写 driver

```
package com.bigdata.mapreduce.sort;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class FlowCountSortDriver {

    public static void main(String[] args) throws ClassNotFoundException, IOException,
        InterruptedException {

        // 1 获取配置信息，或者 job 对象实例
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);

        // 6 指定本程序的 jar 包所在的本地路径
        job.setJarByClass(FlowCountSortDriver.class);

        // 2 指定本业务 job 要使用的 mapper/Reducer 业务类
        job.setMapperClass(FlowCountSortMapper.class);
        job.setReducerClass(FlowCountSortReducer.class);
    }
}
```

```

// 3 指定 mapper 输出数据的 kv 类型
job.setMapOutputKeyClass(FlowBean.class);
job.setMapOutputValueClass(Text.class);

// 4 指定最终输出的数据的 kv 类型
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(FlowBean.class);

// 5 指定 job 的输入原始文件所在目录
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

// 7 将 job 中配置的相关参数, 以及 job 所用的 java 类所在的 jar 包, 提交给 yarn 去运行
boolean result = job.waitForCompletion(true);
System.exit(result ? 0 : 1);
}
}

```

```

root@node1:~# hadoop jar sort-1.0-SNAPSHOT.jar com.bigdata.mapreduce.sort.FlowCountSortDriver /output-Flowsun3 /output-sort4
2024-06-20 16:12:34,491 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java class
es where applicable
2024-06-20 16:12:34,903 INFO client.DefaultHARMPFailoverProxyProvider: Connecting to ResourceManager at node1/10.0.4.11:8032
2024-06-20 16:12:35,155 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool inter
face and execute your application with ToolRunner to remedy this.
2024-06-20 16:12:35,170 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/
job_1718175323880_0014
2024-06-20 16:12:35,402 INFO input.FileInputFormat: Total input files to process : 1
2024-06-20 16:12:35,502 INFO mapreduce.JobSubmitter: number of splits:1
2024-06-20 16:12:35,672 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1718175323880_0014
2024-06-20 16:12:35,672 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-06-20 16:12:35,797 INFO conf.Configuration: resource-types.xml not found
2024-06-20 16:12:35,797 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-06-20 16:12:35,846 INFO impl.YarnClientImpl: Submitted application application_1718175323880_0014
2024-06-20 16:12:35,871 INFO mapreduce.Job: The url to track the job: http://node1:8088/proxy/application_1718175323880_0014/
2024-06-20 16:12:35,872 INFO mapreduce.Job: Running job: job_1718175323880_0014
2024-06-20 16:12:39,947 INFO mapreduce.Job: Job job_1718175323880_0014 running in uber mode : false
2024-06-20 16:12:39,949 INFO mapreduce.Job: map 0% reduce 0%
2024-06-20 16:12:44,027 INFO mapreduce.Job: map 100% reduce 0%
2024-06-20 16:12:48,098 INFO mapreduce.Job: map 100% reduce 100%
2024-06-20 16:12:48,130 INFO mapreduce.Job: Job job_1718175323880_0014 completed successfully
2024-06-20 16:12:48,215 INFO mapreduce.Job: Counters: 54

```

辅助排序和二次排序

有如下订单数据

订单 id 商品 id 成交金额

0000001 Pdt_01 222.8

0000001 Pdt_06 25.8

0000002 Pdt_03 522.8

0000002 Pdt_04 122.4

0000002 Pdt_05 722.4

0000003 Pdt_01 222.8

0000003 Pdt_02 33.8

现在要求出每一个订单中最贵的商品。

(1) 利用“订单 id 和成交金额”作为 key, 可以将 map 阶段读取到的所有订单数据按照 id 分区, 按照金额排序, 发送到 reduce。

(2) 在 reduce 端利用 groupingcomparator 将订单 id 相同的 kv 聚合成组，然后取第一个即是最大值。

代码实现

```
package com.bigdata.mapreduce.order;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.WritableComparable;

public class OrderBean implements WritableComparable<OrderBean> {

    private int order_id; // 订单 id 号
    private double price; // 价格

    public OrderBean() {
        super();
    }

    public OrderBean(int order_id, double price) {
        super();
        this.order_id = order_id;
        this.price = price;
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeInt(order_id);
        out.writeDouble(price);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        order_id = in.readInt();
        price = in.readDouble();
    }

    @Override
    public String toString() {
        return order_id + "\t" + price;
    }

    public int getOrder_id() {
        return order_id;
    }
}
```

```

    public void setOrder_id(int order_id) {
        this.order_id = order_id;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    // 二次排序
    @Override
    public int compareTo(OrderBean o) {

        int result;

        if (order_id > o.getOrder_id()) {
            result = 1;
        } else if (order_id < o.getOrder_id()) {
            result = -1;
        } else {
            // 价格倒序排序
            result = price > o.getPrice() ? -1 : 1;
        }

        return result;
    }
}

package com.bigdata.mapreduce.order;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class OrderDriver {

    public static void main(String[] args) throws Exception, IOException {

        // 1 获取配置信息
        Configuration conf = new Configuration();

```



```

    Job job = Job.getInstance(conf);

    // 2 设置 jar 包加载路径
    job.setJarByClass(OrderDriver.class);

    // 3 加载 map/reduce 类
    job.setMapperClass(OrderMapper.class);
    job.setReducerClass(OrderReducer.class);

    // 4 设置 map 输出数据 key 和 value 类型
    job.setMapOutputKeyClass(OrderBean.class);
    job.setMapOutputValueClass(NullWritable.class);

    // 5 设置最终输出数据的 key 和 value 类型
    job.setOutputKeyClass(OrderBean.class);
    job.setOutputValueClass(NullWritable.class);

    // 6 设置输入数据和输出数据路径
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    // 10 设置 reduce 端的分组
    job.setGroupingComparatorClass(OrderGroupingComparator.class);

    // 7 设置分区
    job.setPartitionerClass(OrderPartitioner.class);

    // 8 设置 reduce 个数
    job.setNumReduceTasks(3);

    // 9 提交
    boolean result = job.waitForCompletion(true);
    System.exit(result ? 0 : 1);
}
}

package com.bigdata.mapreduce.order;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class OrderGroupingComparator extends WritableComparator {

    protected OrderGroupingComparator() {
        super(OrderBean.class, true);
    }
}

```

```

@SuppressWarnings("rawtypes")
@Override
public int compare(WritableComparable a, WritableComparable b) {

    OrderBean aBean = (OrderBean) a;
    OrderBean bBean = (OrderBean) b;

    int result;
    if (aBean.getOrder_id() > bBean.getOrder_id()) {
        result = 1;
    } else if (aBean.getOrder_id() < bBean.getOrder_id()) {
        result = -1;
    } else {
        result = 0;
    }

    return result;
}
}

package com.bigdata.mapreduce.order;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class OrderMapper extends Mapper<LongWritable, Text, OrderBean, NullWritable> {
    OrderBean k = new OrderBean();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {

        // 1 获取一行
        String line = value.toString();

        // 2 截取
        String[] fields = line.split("\t");

        // 3 封装对象
        k.setOrder_id(Integer.parseInt(fields[0]));
        k.setPrice(Double.parseDouble(fields[2]));

        // 4 写出
        context.write(k, NullWritable.get());
    }
}

```

```

    }
}
package com.bigdata.mapreduce.order;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Partitioner;

public class OrderPartitioner extends Partitioner<OrderBean, NullWritable> {

    @Override
    public int getPartition(OrderBean key, NullWritable value, int numReduceTasks) {

        return (key.getOrder_id() & Integer.MAX_VALUE) % numReduceTasks;
    }
}

package com.bigdata.mapreduce.order;
import java.io.IOException;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class OrderReducer extends Reducer<OrderBean, NullWritable, OrderBean,
NullWritable> {









    @Override
    protected void reduce(OrderBean key, Iterable<NullWritable> values, Context context)
        throws IOException, InterruptedException {

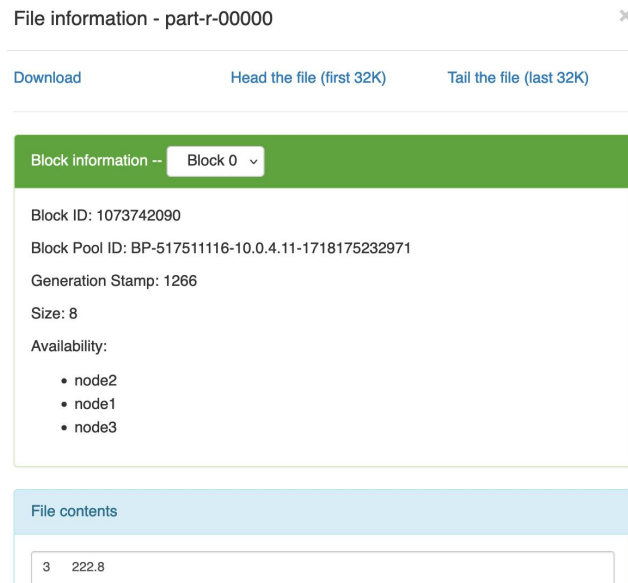
        context.write(key, NullWritable.get());
    }
}

```

运行:

hadoop jar order-1.0-SNAPSHOT.jar com.bigdata.mapreduce.order.OrderDriver /order/input /order/output

<input type="text"/>	Go!				
Search: <input type="text"/>					
Replication	Block Size	Name			
3	128 MB	_SUCCESS			
3	128 MB	part-r-00000			
3	128 MB	part-r-00001			
3	128 MB	part-r-00002			
<div> Previous 1 Next </div>					



(9) Hive 实操

安装 MySQL

1. 安装 MySQL 服务器

在 node1 上执行以下命令:

```
sudo apt update
```

```
sudo apt-get install mysql-server
```

2. 配置 MySQL

启动 MySQL 服务并进行安全设置:

```
sudo systemctl start mysql
```

```
sudo mysql_secure_installation
```

按照提示完成安全设置, 创建 root 用户密码等。

3. 创建 Hive 数据库

使用 MySQL 登录并创建 Hive 所需的数据库和用户:

```
sudo mysql -u root -p
```

在 MySQL shell 中执行以下命令:

```
root@node1:/usr/local/hadoop# systemctl status mysql
● mysql.service - MySQL Community Server
   Loaded: loaded (/usr/lib/systemd/system/mysql.service; enabled; preset: enabled)
   Active: active (running) since Sun 2024-06-23 07:09:43 UTC; 3h 12min ago
     Process: 3251 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
    Main PID: 3259 (mysqld)
      Status: "Server is operational"
        Tasks: 86 (limit: 4549)
       Memory: 434.7M (peak: 435.8M)
          CPU: 4min 59.309s
         CGroup: /system.slice/mysql.service
                 └─3259 /usr/sbin/mysqld
```

安装 Hive

1. 下载并解压 Hive

在 node1 上执行以下命令来下载并解压 Hive:

```
cd /usr/local
wget https://downloads.apache.org/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz
sudo tar -zxvf apache-hive-3.1.3-bin.tar.gz
sudo mv apache-hive-3.1.3-bin hive
rm apache-hive-3.1.3-bin.tar.gz
```

2. 配置环境变量

在~/.bashrc 文件中添加以下内容:

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
```

使更改生效:

```
source ~/.bashrc
```

```
export JAVA_HOME=/usr/lib/jvm/temurin-8-jdk-arm64
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
```

3. 配置 Hive

创建 Hive 配置目录并编辑配置文件:

```
sudo mkdir -p $HIVE_HOME/conf
cd $HIVE_HOME/conf
# sudo cp $HIVE_HOME/conf/hive-default.xml.template $HIVE_HOME/conf/hive-site.xml
sudo vim $HIVE_HOME/conf/hive-site.xml
```

'hive-site.xml':

```
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost/metastore</value>
    <description>JDBC connect string for a JDBC metastore</description>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
    <description>Driver class name for a JDBC metastore</description>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hiveuser</value>
    <description>Username to use against metastore database</description>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hivepassword</value>
    <description>Password to use against metastore database</description>
  </property>
```

```

<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/user/hive/warehouse</value>
  <description>location of default database for the warehouse</description>
</property>
<property>
  <name>hive.cli.print.header</name>
  <value>true</value>
</property>
<property>
  <name>hive.cli.print.current.db</name>
  <value>true</value>
</property>
</configuration>

```

```

<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost/metastore</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.cj.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hiveuser</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hivepassword</value>
  </property>
  <property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
    <description>location of default database for the warehouse</description>
  </property>
  <property>
    <name>hive.cli.print.header</name>
    <value>true</value>
  </property>
  <property>
    <name>hive.cli.print.current.db</name>
    <value>true</value>
  </property>
</configuration>

```

4. 下载 MySQL JDBC 驱动

下载 MySQL JDBC 驱动并放到 Hive 的 lib 目录:

```
cd /usr/local
```

```
wget
```

```
https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.26.tar.gz
```

```
tar -zxvf mysql-connector-java-8.0.26.tar.gz
```

```
sudo cp mysql-connector-java-8.0.26/mysql-connector-java-8.0.26.jar $HIVE_HOME/lib/
```

```
rm mysql-connector-java-8.0.26.tar.gz
```

```
# rm -r mysql-connector-java-8.0.26
```

5. 初始化 Hive Metastore

运行以下命令初始化 Hive Metastore:

```
schematool -initSchema -dbType mysql
```

6. 验证安装

启动 Hive CLI 并验证安装:

```
hive
```

```
root@node1:/usr/local/hadoop# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = dc98435c-f942-4185-8153-036120d93e9c

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-3.1.3.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Hive Session ID = 08f650c7-1a05-45ea-b56c-87c4fa487da2
```

7. 修改 HDFS 权限

```
hdfs dfs -mkdir -p /user/hive/warehouse
```

```
hdfs dfs -chmod -R 777 /user/hive/warehouse
```

验证 Hive

1. 创建一个目录来存放数据文件

```
hdfs dfs -mkdir -p /user/hive/warehouse/user_data
```

2. 上传数据文件到新目录

创建一个数据文件:

```
echo -e "1,John\n2,Jane\n3,Bob" > data.csv
```

然后将数据文件上传到 HDFS 的 `/user/hive/warehouse/user_data` 目录:

```
hdfs dfs -put data.csv /user/hive/warehouse/user_data/
```

3. 在 Hive CLI 中, 运行以下命令创建一个示例表并查询:

```
CREATE EXTERNAL TABLE IF NOT EXISTS user_data (
  id INT,
  name STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs://node1:8020/user/hive/warehouse/user_data';

SELECT * FROM user_data;
```

```
hive> SELECT * FROM user_data;
OK
1      John
2      Jane
3      Bob
Time taken: 0.83 seconds, Fetched: 3 row(s)
```

外部表 案例实操

分别创建部门和员工外部表，并向表中导入数据。

1 原始数据

数据以制表符分隔

dept.txt

```
10      ACCOUNTING      1700
20      RESEARCH1800
30      SALES    1900
40      OPERATIONS      1700
```

emp.txt

```
7369    SMITH    CLERK    7902    1980-12-17    800.00    0.00    20
7499    ALLEN    SALESMAN 7698    1981-2-20    1600.00    300.00    30
7521    WARD    SALESMAN 7698    1981-2-22    1250.00    500.00    30
7566    JONES    MANAGER  7839    1981-4-22 2975.00 0.00    20
7654    MARTIN    SALESMAN 7698    1981-9-28    1250.00    1400.00    30
7698    BLAKE    MANAGER  7839    1981-5-12 850.00 0.00    30
7782    CLARK    MANAGER  7839    1981-6-9 2450.00 0.00    10
7788    SCOTT    ANALYST  7566    1987-4-19    3000.00    0.00    20
7839    KING    PRESIDENT    NULL    1981-11-17    5000.00    0.00    10
7844    TURNER    SALESMAN 7698    1981-9-8 1500.00 0.00    30
7876    ADAMS    CLERK    7788    1987-5-23    1100.00    0.00    20
7900    JAMES    CLERK    7698    1981-12-3    950.00    0.00    30
7902    FORD    ANALYST  7566    1981-12-3    3000.00    0.00    20
7934    MILLER    CLERK    7782    1982-1-23    1300.00    0.00    10
```

2 建表语句

创建部门表

```
create external table if not exists dept(
deptno int,
dname string,
loc int
)
row format delimited fields terminated by '\t';
```

创建员工表

```
create external table if not exists emp(
empno int,
ename string,
```



```
job string,  
mgr int,  
hiredate string,  
sal double,  
comm double,  
deptno int)  
row format delimited fields terminated by '\t';
```

3 查看创建的表

```
show tables;
```

```
hive (default)> create external table if not exists dept(  
    > deptno int,  
    > dname string,  
    > loc int  
    > )  
    > row format delimited fields terminated by '\t';  
OK  
Time taken: 1.022 seconds  
hive (default)> create external table if not exists emp(  
    > empno int,  
    > ename string,  
    > job string,  
    > mgr int,  
    > hiredate string,  
    > sal double,  
    > comm double,  
    > deptno int)  
    > row format delimited fields terminated by '\t';  
OK  
Time taken: 0.103 seconds  
hive (default)> show tables;  
OK  
tab_name  
dept  
emp  
test_table  
user_data  
Time taken: 0.149 seconds, Fetched: 4 row(s)
```

4 向外部表中导入数据

导入数据，数据文件位于虚拟机上

```
load data local inpath '/root/dept.txt' into table default.dept;
```

```
load data local inpath '/root/emp.txt' into table default.emp;
```

```
hive (default)> load data local inpath '/root/dept.txt' into table default.dept;  
Loading data to table default.dept  
OK  
Time taken: 0.761 seconds  
hive (default)> load data local inpath '/root/emp.txt' into table default.emp;  
Loading data to table default.emp  
OK  
Time taken: 0.292 seconds
```

查询结果

```
select * from dept;
```

```
select * from emp;
```

```
hive (default)> select * from emp;
OK
emp.empno      emp.ename      emp.job emp.mgr emp.hiredate      emp.sal emp.comm      emp.deptno
7369 SMITH      CLERK  7902   1980-12-17      800.0  0.0    20
7499 ALLEN      SALESMAN 7698   1981-2-20      1600.0 300.0  30
7521 WARD      SALESMAN 7698   1981-2-22      1250.0 500.0  30
7566 JONES      MANAGER 7839   1981-4-2        2975.0 0.0    20
7654 MARTIN     SALESMAN 7698   1981-9-28      1250.0 1400.0 30
7698 BLAKE      MANAGER 7839   1981-5-1        2850.0 0.0    30
7782 CLARK      MANAGER 7839   1981-6-9        2450.0 0.0    10
7788 SCOTT      ANALYST 7566   1987-4-19      3000.0 0.0    20
7839 KING      PRESIDENT NULL   1981-11-17      5000.0 0.0    10
7844 TURNER     SALESMAN 7698   1981-9-8        1500.0 0.0    30
7876 ADAMS      CLERK  7788   1987-5-23      1100.0 0.0    20
7900 JAMES      CLERK  7698   1981-12-3      950.0  0.0    30
7902 FORD      ANALYST 7566   1981-12-3      3000.0 0.0    20
7934 MILLER     CLERK  7782   1982-1-23      1300.0 0.0    10
Time taken: 0.141 seconds, Fetched: 14 row(s)
```

5 查看表格式化数据

```
desc formatted dept;
```

```
hive (default)> desc formatted dept;
OK
col_name      data_type      comment
# col_name      data_type      comment
deptno        int
dname         string
loc           int

# Detailed Table Information
Database:      default
OwnerType:     USER
Owner:         root
CreateTime:    Sun Jun 23 09:23:33 UTC 2024
LastAccessTime: UNKNOWN
Retention:     0
Location:      hdfs://node1:8020/user/hive/warehouse/dept
Table Type:    EXTERNAL_TABLE
Table Parameters:
  EXTERNAL          TRUE
  bucketing_version 2
  numFiles          1
  numRows           0
  rawDataSize       0
  totalSize         69
  transient_lastDdlTime 1719134695

# Storage Information
SerDe Library:  org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:    org.apache.hadoop.mapred.TextInputFormat
OutputFormat:   org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:     No
Num Buckets:    -1
Bucket Columns: []
Sort Columns:   []
Storage Desc Params:
  field.delim      \t
  serialization.format \t
Time taken: 0.223 seconds, Fetched: 34 row(s)
```

(二) 主要收获与体会

在此次大数据应用项目实践中，我获得了许多宝贵的收获与体会。通过实践，我对大数据技术的核心组件有了深入的了解。在 Hadoop 生态系统中，如 HDFS、YARN 和 MapReduce 等关键技术的工作原理和实现细节，通过实操和代码实现，我从理论理解走向了实际操作，极大地提升了自己的技术水平。通过配置 HDFS 集群环境，我深刻理解了 NameNode 和 DataNode 的协同工作机制，这是对其在实际应用中重要性的切身体会。

本次项目实践让我深刻认识到数据处理过程中的优化技巧和最佳实践。数据布局 and 分区的优化、合理的

MapReduce 作业参数配置、以及 Mapper 和 Reducer 实现的优化，这些都直接影响到大数据处理的效率和性能。在实际操作中，通过对这些方面的优化实践，我学会了如何通过调整作业的分片数量、合理配置任务资源等方法，来提升数据处理的效率，并有效地避免了数据倾斜和资源浪费的问题。

通过对 MapReduce 编程模型的深度实践，我加深了对分布式计算思想的理解。在实际的 WordCount 案例中，从数据准备、编写 Mapper 类和 Reducer 类，到最后的集群测试，我将理论知识应用到实际编程中去。这个过程中，锻炼了我的编程能力，让我体会到分布式计算在处理大规模数据时的强大优势和复杂性。

通过这次实践，我深刻体会到了不断学习和更新知识的重要性。大数据技术发展迅速，新技术和新工具层出不穷。通过此次项目实践，我意识到必须保持持续学习的态度，不断更新自己的知识储备，才能在快速变化的技术环境中保持竞争力。这次大数据应用项目实践让我在技术层面上收获颇丰，在实践过程中提升了团队合作和项目管理的能力。这些宝贵的收获和体会将为我未来的学习和职业发展奠定坚实的基础。

(三) 实践成果

(1) Hadoop 集群

在此次实践中，我成功搭建了一个功能完善的 Hadoop 集群，包括 HDFS、YARN 和 MapReduce 组件。这一过程让我深入了解了 Hadoop 生态系统的核心组件及其工作原理，提升了我的实际操作能力和解决问题的能力。

1 环境搭建

在三台虚拟机上分别安装了 Debian 12 操作系统，并通过配置静态 IP 地址和启用 SSH 服务，确保了虚拟机之间的网络通信畅通。在所有虚拟机上安装了 JDK，是 Hadoop 运行的基础环境。通过配置环境变量 JAVA_HOME 和 HADOOP_HOME，确保 Hadoop 可以正确识别和使用 JDK。

```
vps@01281:~$ hostname -I
192.168.137.21
vps@01281:~$ ping 192.168.137.22
PING 192.168.137.22 (192.168.137.22) 56(84) bytes of data.
 64 bytes from 192.168.137.22: icmp_seq=1 ttl=64 time=0.436 ms
 64 bytes from 192.168.137.22: icmp_seq=2 ttl=64 time=0.254 ms
 64 bytes from 192.168.137.22: icmp_seq=3 ttl=64 time=0.362 ms
 64 bytes from 192.168.137.22: icmp_seq=4 ttl=64 time=0.293 ms
^C
--- 192.168.137.22 ping statistics ---
 4 packets transmitted, 4 received, 0% packet loss, time 3064ms
 rtt min/avg/max/mdev = 0.254/0.336/0.436/0.069 ms
vps@01281:~$ ping 192.168.137.23
PING 192.168.137.23 (192.168.137.23) 56(84) bytes of data.
 64 bytes from 192.168.137.23: icmp_seq=1 ttl=64 time=0.652 ms
 64 bytes from 192.168.137.23: icmp_seq=2 ttl=64 time=0.233 ms
 64 bytes from 192.168.137.23: icmp_seq=3 ttl=64 time=0.294 ms
 64 bytes from 192.168.137.23: icmp_seq=4 ttl=64 time=0.473 ms
^C
--- 192.168.137.23 ping statistics ---
 4 packets transmitted, 4 received, 0% packet loss, time 3051ms
 rtt min/avg/max/mdev = 0.233/0.413/0.652/0.163 ms
vps@01281:~$
```

2 HDFS 配置

在 HDFS 配置方面，在每台虚拟机上安装了 Hadoop，并配置了核心文件 core-site.xml 和 hdfs-site.xml。在 core-site.xml 中指定 HDFS 的默认文件系统，配置 Hadoop 临时目录。在 hdfs-site.xml 中设置数据块的副本数量，确保数据的高可靠性。在 NameNode 上执行了格式化操作，初始化 HDFS 文件系统。

```
<configuration>
  <!-- 默认文件系统的名称。通过URI中schema区分不同文件系统 -->
  <!-- file://本地文件系统 hdfs://hadoop分布式文件系统 -->
  <!-- gfs://google文件系统 -->
  <!-- hdfs文件系统访问地址: http://node1:8020 -->
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://node1:8020</value>
  </property>

  <!-- 设置Hadoop本地保存数据路径 -->
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/hadoop/tmp</value>
  </property>

  <!-- 设置HDFS web UI用户身份 -->
  <property>
    <name>hadoop.http.staticuser.user</name>
    <value>root</value>
  </property>
</configuration>
```

3 NameNode 和 DataNode 配置

NameNode 是 HDFS 的核心，负责管理文件系统的命名空间和元数据。在实践中，在主节点上配置 NameNode，并确保其高可用性。设置 SecondaryNameNode，用于定期合并元数据快照和编辑日志，减轻 NameNode 的负担。DataNode 是实际存储数据块的节点，在其他虚拟机上配置 DataNode，确保数据块在多个节点上均匀分布，提高数据的容错性和可用性。

4 YARN 配置

YARN 是 Hadoop 的资源管理和作业调度框架。在 YARN 配置中，在主节点上配置 ResourceManager，在所有节点上配置了 NodeManager。在 yarn-site.xml 中指定 ResourceManager 的主机名和端口和 NodeManager 的附加服务。通过这些配置，YARN 可以高效管理和调度集群资源，确保作业的顺利执行。

```
<configuration>

  <!-- Site specific YARN configuration properties -->
  <!-- 设置YARN集群角色运行机器位置 -->
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>node1</value>
  </property>

  <!-- ModeManager上运行的附属服务，需配置成mapreduce_shuffle才可运行程序。 -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

5 启动和验证

完成所有配置后，依次启动了 HDFS 和 YARN 集群。在 NameNode 上执行 start-dfs.sh 命令启动 HDFS 集群，执行 start-yarn.sh 命令启动 YARN 集群。启动完成后通过 Web UI 界面验证了集群的健康状态。在 HDFS 的 Web UI 上查看集群的存储使用情况和数据块分布。在 YARN 的 Web UI 上，监控资源的使用情况和作业的运行状态。

Overview 'node1:8020' (✓active)

Started:	Sun Jun 23 14:57:32 +0800 2024
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 16:22:00 +0800 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-201362dd-09ad-4ede-8191-b72f8494b190
Block Pool ID:	BP-517511116-10.0.4.11-1718175232971

6 集群优化与维护

在实践过程中学习如何优化 Hadoop 集群的性能。通过调整数据块大小和副本数量，优化数据分布策略，提升了 HDFS 的读写性能。学习如何监控集群的运行状态，使用工具如 Ganglia 和 Nagios 来实时监控集群的资源使用情况和节点状态。通过这些工具可以及时发现解决集群运行中的问题，确保集群的高可用性和可靠性。

(2) MapReduce Java 编程

在 MapReduce 编程部分，我完成了多个案例的实操，包括经典的 WordCount 案例、序列化案例、Partition 分区案例、WritableComparable 排序案例、Reduce join 案例、辅助排序和二次排序案例。在这些案例中，我不仅掌握了 MapReduce 编程模型的基本原理，还深入学习了数据分片、任务调度、数据序列化与反序列化等关键技术。特别是在 WordCount 案例中，我从数据准备、编写 Mapper 类和 Reducer 类，到最后的集群测试，完整地经历了一个 MapReduce 作业的开发流程。这些实践让我更深刻地理解了分布式计算的优势和复杂性。

1 WordCount 案例实操

WordCount 是 MapReduce 的经典入门案例，其主要目的是统计文本文件中每个单词的出现次数。在这个案例中，Mapper 类用于将输入的文本文件分割成一个个单词，并输出每个单词的键值对。Reducer 类用于汇总每个单词的出现次数。Driver 类用于配置作业并提交到集群运行。

```
root@node1:~# hadoop jar wordcount-1.0-SNAPSHOT.jar com.bigdata.mapreduce.wordcount.WordcountDriver /wordcount/input /wordcount/output
2024-06-22 10:56:13,392 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2024-06-22 10:56:13,763 INFO client.DefaultHadoopFailoverProxyProvider: Connecting to ResourceManager at node1/10.0.4.11:8032
2024-06-22 10:56:14,073 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2024-06-22 10:56:14,078 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1718175323880_0015
2024-06-22 10:56:14,325 INFO input.FileInputFormat: Total input files to process : 1
2024-06-22 10:56:14,408 INFO mapreduce.JobSubmitter: number of splits:1
2024-06-22 10:56:14,534 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1718175323880_0015
2024-06-22 10:56:14,534 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-06-22 10:56:14,652 INFO conf.Configuration: resource-types.xml not found
2024-06-22 10:56:14,652 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-06-22 10:56:14,710 INFO impl.YarnClientImpl: Submitted application application_1718175323880_0015
2024-06-22 10:56:14,737 INFO mapreduce.Job: The url to track the job: https://node1:8088/proxy/application_1718175323880_0015/
2024-06-22 10:56:14,737 INFO mapreduce.Job: Running job: job_1718175323880_0015
2024-06-22 10:56:19,813 INFO mapreduce.Job: Job job_1718175323880_0015 running in uber mode : false
2024-06-22 10:56:19,815 INFO mapreduce.Job: map 0% reduce 0%
2024-06-22 10:56:23,902 INFO mapreduce.Job: map 100% reduce 0%
2024-06-22 10:56:26,944 INFO mapreduce.Job: map 100% reduce 100%
2024-06-22 10:56:28,002 INFO mapreduce.Job: Job job_1718175323880_0015 completed successfully
2024-06-22 10:56:28,084 INFO mapreduce.Job: Counters: 54
File System Counters
  FILE: Number of bytes read=2864
  FILE: Number of bytes written=558225
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=1589
  HDFS: Number of bytes written=1339
  HDFS: Number of read operations=0
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
  HDFS: Number of bytes read erasure-coded=0
```

2 序列化案例实操

在序列化案例中使用 Hadoop 的 Writable 接口对自定义数据类型进行序列化和反序列化。创建 FlowBean 类，用于记录手机号的上行流量、下行流量和总流量。FlowBean 的 write 和 readFields 方法实现了数据的序列化和反序列化。

File information - part-r-00000

Download

Head the file (first 32K)

Tail the file (last 32K)

Block information --

Block 0

Block ID: 1073742019

Block Pool ID: BP-517511116-10.0.4.11-1718175232971

Generation Stamp: 1195

Size: 104

Availability:

- node2
- node3
- node1

File contents

13560436666	1116	954	2070
13600001111	7282	572	7854
13722223333	2881	828	3709
13822223333	2821	823	3644

3 Partition 分区案例

Partition 分区案例使用自定义 Partitioner 类将 Map 输出的键值对分配到不同的 Reducer 进行处理。自定义 Partitioner 类用于根据手机号的归属地将数据分配到不同的 Reducer 中。

/flowsun.partition/output

Go!

Show

25

entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	0 B	Jun 22 20:32	3	128 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	26 B	Jun 22 20:32	3	128 MB	part-r-00000	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	26 B	Jun 22 20:32	3	128 MB	part-r-00001	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	26 B	Jun 22 20:32	3	128 MB	part-r-00002	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	26 B	Jun 22 20:32	3	128 MB	part-r-00003	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	0 B	Jun 22 20:32	3	128 MB	part-r-00004	

4 WritableComparable 排序案例

WritableComparable 排序案例对 Map 输出的键值对进行排序。自定义的 Key 类实现了 WritableComparable 接口，用于对键值对进行排序。GroupingComparator 类用于在 Reducer 端对键值对进行分组处理。

5 Reduce Join 案例

Reduce join 案例在 MapReduce 作业中实现表连接操作。两个 Mapper 类分别读取订单表和商品表的数据，并输出键为商品 ID 的键值对。Reducer 类将同一商品 ID 的订单数据和商品数据进行连接，并输出完整的订单信息。

File contents

1001	小米	1
1001	小米	1
1002	华为	2
1002	华为	2
1003	格力	3
1003	格力	3

6 辅助排序和二次排序案例

辅助排序和二次排序案例通过自定义 Comparator 类实现复杂的排序逻辑。自定义 Comparator 类用于对 Map 输出的键值对进行二次排序，Mapper 和 Reducer 类处理排序后的数据。

(3) Hive 数据仓库

在 Hive 数据仓库的学习和实践中，通过 MySQL、HDFS 与 Hive 的联合使用，并进行了外部表案例的实操。通过创建部门和员工外部表，并向表中导入数据，在操作过程中，在 Hive 中创建表、导入数据、执行查询以及优化查询性能。配置 Hive 与 HDFS 的联合使用使得数据在分布式存储系统中的管理更加高效。

```
hive (default)> desc formatted dept;
OK
col_name      data_type      comment
# col_name      data_type      comment
deptno        int
dname         string
loc           int

# Detailed Table Information
Database:      default
OwnerType:     USER
Owner:         root
CreateTime:    Sun Jun 23 09:23:33 UTC 2024
LastAccessTime: UNKNOWN
Retention:     0
Location:      hdfs://node1:8020/user/hive/warehouse/dept
Table Type:    EXTERNAL_TABLE
Table Parameters:
EXTERNAL              TRUE
bucketing_version    2
numFiles              1
numRows              0
rawDataSize          0
totalSize            69
transient_lastDdlTime 1719134695
```

