一个提高气象数值模式计算性能的并行I/O优化技术

邓平 张庆花 张玉龙

阐述了开展并行I/O研究对气象数值预报的重要意义，以及通过并行I/O技术充分发挥并行存储性能的必要性和迫切性，提出了并行文件系统底层优化策略及基于MPI-IO的高层I/O库的优化方法，优化后可支持气象要素的并行输出，试验测试数据表明，优化后输出效率明显提升。

DOI：10.3969/j.issn.2095-1973.2019.03.033

气象数值模式大规模并行计算会在短时间内产生大量的数据，并且小文件数量巨大，对整个系统的I/O性能、稳定性要求很高，一般要求有分布式存储系统或者并行文件系统，否则容易形成I/O瓶颈，很难获得理想的模式计算加速比。随着气象数值模式的时空分辨率不断提升，要求更大的数据存储容量及更高的并发I/O存储带宽，I/O性能成为模式运行效率的主要瓶颈之一。本文以深圳市气象局逐时雷达同化预报系统（Hourly Assimilation Prediction System，HAPS）1 km算例为例进行分析。

逐时雷达同化预报系统是深圳市气象局业务化运行系统，“十三五”期间，为满足精细化城市预报需求，该系统的水平分辨率要从4 km提升至1 km，垂直分辨率要从51层提高到70层。升级后的应用系统对整个系统的I/O性能、稳定性要求很高，模式读入输入数据后，以特定积分步长、完成特定的计算任务，在特定的时间间隔把计算结果的输出到文件，该模式的输出文件为NetCDF数据格式。

HAPS 1 km算例模拟时间范围为24 h，每积分30 min输出一个wrfout文件，每个wrfout文件大小为8.1 G，一次算例的输出文件大小共为396.9 G，串行输出时耗时约为5880 s。因此，传统的串行输出方式一方面影响了模式运行的效率，另一方面造成了计算资源的浪费，且未能充分发挥并行存储的优势。

针对深圳市气象局数值计算时普遍遇到的I/O性能瓶颈问题，开展了面向气象领域的数据并行处理技术的研究，以解决模式运行时数据的读写效率问题。经过深入分析，可从以下两个方面对应用程序的输出效率进行优化。

一方面，从存取文件的方式以及应用程序内部的读写算法进行调整，提高气象应用的文件访问效率。应用并行I/O库，由同一并发程序的多个进程产生对文件数据的并发请求。减少了串行读取数据时并行通信域中其他处理器核心处于等待状态造成的计算资源浪费，同时多个处理器并行访问数据可充分发挥并行存储的性能优势。另一方面，从文件系统角度，针对模式计算和前后处理的I/O特征，进行并行文件系统底层软件优化和I/O策略优化。优化策略主要包含锁颗粒优化、atime优化、文件锁优化、条带块大小自适应、条带宽度调整、条带预读、数据均衡、基于智能算法提高缓存命中率，优化并行文件系统的性能。

1 并行I/O优化

读写文件策略（图1）分为三类：

1）只有一个进程读写

主进程需要负责所有文件的I/O操作，将文件中的所有数据读入自己的缓冲区（buffer），然后用MPI发送接收函数将大部分数据传给并行程序的所有任务。计算结束后，主进程负责将所有结果数据写到文件。显然，该策略时，应用存取数据的效率取决于

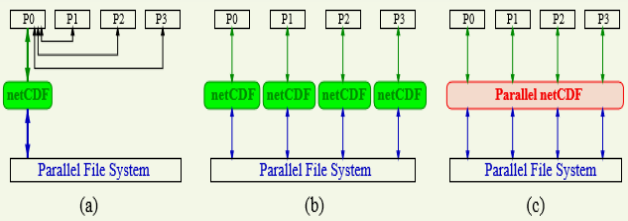

图1 I/O策略

收稿日期：2018年11月29日；修回日期：2019年3月29日

MPI通信性能和主任务对文件的访问性能。

2）多个进程分别读写

用户将应用所需的一个数据文件按照特定的方式分成多个文件，每个进程只操作自己的文件，彼此间不协调，相互独立。这种策略优点为既能同时使用计算服务器的多个网络通道，发挥并行存储系统的多客户端接入能力，并且没有额外的锁开销，适用于进程数量较少且单进程访存的数据量大的情况。缺点是当进程数量较多时，输出的文件数据太多，增加了后续处理的复杂度及难度。

3）多个进程读写同一文件

MPI-2对MPI-1进程了几个重要的扩展，增加了对MPI-IO的支持，MPI-IO是一个并行I/O库，提供了执行可移植的、用户级的I/O 操作接口。MPI可以通过该接口在文件和进程间传送数据。从而实现了并行程序通过多任务并发读取同一个文件。多个进程相互配合，避免无用操作。

MPI-IO分为非集中式I/O操作及集中式I/O操作，相比于POSIX I/O接口，其具有以下优点：1）更高级别的数据接口，可以并行写入复杂的数据类型，允许并行文件系统及MPI-IO调用优化性能；2）支持数据一致性和原子性；3）优化I/O函数。目前，已有Parallel-NetCDF及HDF5等支持MPI-IO的专业库。Parallel-NetCDF继承了传统的NetCDF代码。保持了NetCDF数据格式的兼容性的特点，是一个高性能的对NetCDF格式数据进行访问的应用编程接口。Parallel-NetCDF及HDF5均创建于MPI-IO上层，提供集合I/O，驱动底层MPI-IO，获得持续的并行访问性能。可极大地提高I/O效率。

原HAPS模式I/O策略采用串行I/O方式，只有一个进程进行读写，主进程负责所有文件的I/O操作，数据输出效率成为模式整体效率的瓶颈。而多个进程分别读写的方式则会产生与进程数相同的文件，输出的文件数目很多，增加了后处理的复杂度及难度。因此本文研究在HAPS模式中应用并行I/O库，使用多个进程读写同一个文件的策略，一方面通过多进程并行输出提升I/O的性能，另一方面采用该策略不会增加后处理的复杂度。

2 文件系统优化

MPI-IO基于软件包ROMIO实现，包含多个驱动，并可针对文件系统进行针对性优化，目前MPI-IO的优化技术在部分文件系统的表现不如人意，根源是部分文件系统的分布式锁没有配合好。目前锁协议类型多，通过研究测试选择合适的锁协议，减少申请锁的次数，减少不必要的时间消耗。

分布式锁是分布式系统控制同步访问共享资源的一种方式。如果不同的系统或是同一个系统的不同主机之间共享了一个或一组资源，那么访问这些资源的时候，需要互斥来防止彼此干扰来保证一致性，在这种情况下，便需要使用到分布式锁。对于文件系统，资源可以是文件的数据区间，也可以是文件的元数据属性；对于数据库系统，资源可以是一张表或者一条记录。资源实体是描述资源各种属性的实体结构。

文件系统根据共享资源不同支持多种粒度的锁：节点锁、文件系统锁、目录锁、文件锁和区间锁等。通过细化分布式锁的粒度降低应用请求之间的冲突，提升系统的并发性能。

根据锁处理过程中的角色不同，存储文件系统将系统中的逻辑节点划分为协调者和发起者。协调者负责某一资源上的所有锁的授予召回操作，协调者根据资源上不同类型锁之间的优先级和互斥性决定是否授予相应的锁。每个资源对应一个协调者，但是一个协调者可以管理多个资源的锁操作。文件系统中存在多个协调者，而且一个资源可以切换协调者，但是同一时间，一个资源只有一个协调者。锁的发起者可以是客户端节点的应用进程，也可以是存储后台服务进程，同一资源的锁发起者之间存在竞争。

2.1 锁颗粒优化

确定加锁机制后，对文件系统中选择合适的锁颗粒，不合适的锁颗粒就会造成巨大的资源消耗：内存、CPU计算能力、存储空间等。通过并行文件系统测试优化锁颗粒的设置。

文件系统的默认配置一般针对顺序大块I/O场景性能最优，通过锁扩展提升锁命中度，进一步提升I/O性能。多进程并发读写同一个文件的场景，锁扩展后每个进程获取到更大范围的锁空间，导致进程之间锁冲突的概率增大。对锁冲突的处理会降低I/O效率，关闭锁扩展，根据I/O粒度加对应的锁，降低锁冲突可以极大的提升I/O性能。

2.2 atime 优化

按照标准的POSIX语义，读操作需要更新文件的atime。在业务运行过程中，读取输入数据阶段，多个进程读同一个文件的场景，即为多个进程并发修改同一个文件的atime的场景。

对于需要上万核心并发执行的高性能作业，对于输入文件atime的修改会成为读取输入数据的瓶颈。部分文件系统支持挂载时noatime参数，即关闭atime更新，提升读取输入数据性能，关闭atime的优化方案适

用于不依赖atime的应用程序。对于依赖atime的应用程序，文件系统通过细化元数据锁粒度，设置单独的atime锁，提升atime更新性能。

2.3 文件锁优化

应用程序必须调用MPI库读写数据的场景，文件系统通过提升锁服务线程个数并且在客户端开启锁缓存的方式提升加锁性能。

文件系统将用户文件划分成定长的数据段，每个数据段中的数据对象以条带化的方式存放在多个存储设备上。为了保证数据的可靠性，同一个数据段中的数据对象之间采用多副本或纠删码的方式进行冗余保护：当部分数据对象所在的存储节点、网络或存储设备发生故障时，系统可以通过其他数据对象来及时重建该数据，以保证数据的可靠性。

文件系统对I/O性能的优化，除了锁颗粒、atime、文件锁优化之外，还包括以下几个方面：

1）条带块大小自适应：针对不同大小的文件，文件系统可以自适应的选择不同的条带块大小，确保各种大小文件的I/O性能达到最优；

2）增加条带宽度：文件系统在线调整条带宽度，在存储系统扩容后，可以通过增加条带宽度，优化文件的顺序访问I/O性能；

3）条带预读：条带预读是文件系统预读算法中的一种，通过条带预读可以进一步优化顺序读I/O的性能；

4）数据均衡：文件系统可存储节点和存储设备的容量进行数据均衡，在保证冗余规则的前提下，调整数据对象在系统中的分布情况，提高数据访问性能。

条带化主要受限于存储集群规模和数据冗余度。更大规模的集群可以支持更多的条带化配置和数据冗余度配置，I/O性能优化的空间越大。

2.4 缓存命中率优化

随着IT技术的发展，CPU计算能力发展迅速，机械硬盘（HDD）容量虽有着惊人的扩展，但是性能的提升远远不够，硬盘与CPU之间的性能鸿沟越来越大。在工作负载较大、对性能要求较高的I/O中，硬盘的性能往往成为瓶颈。增加内存（RAM Cache）可以明显提升系统的访问速度，但单位容量的内存价格过于昂贵，绝大多数的用户难以接受。

SSD相较于机械硬盘，带宽及响应时间有着明显的优势，并且容量远大于普通内存。将SSD作为缓存资源，可以显著降低存储系统的响应时间，有效提高数据的访问频率。

文件系统利用SSD盘对随机小文件读取速度快的特点，将SSD盘组成介于HDD与内存之间的二级缓存池。通过智能算法将访问频度高的随机小文件热点数据存放到SSD上，应用程序再次访问该数据时，可以直接从SSD上获取。由于SSD盘的数据读取速度远远高于机械硬盘，因此可以显著缩短热点数据的响应时间，从而提升系统的性能。

存储系统的数据节点上配置一块或多块的SSD，通过配置参数设定作为缓存资源的SSD容量（其余容量可以用作普通数据盘）。SSD只能加速所在节点上的数据访问，SSD故障后不影响数据正确性，拔掉SSD缓存盘后再插入，其上的缓存数据会自动清除。

SSD缓存功能需要消耗一定的内存容量来记录数据的访问热度等信息。一般情况下，开启SSD缓存功能后，消耗的内存与SSD缓存容量比例为1∶50，即1 TB的SSD缓存容量需要消耗20GB的普通内存。

基于智能算法，自动识别连续/随机I/O，过滤大块连续I/O操作，仅将随机访问的小块数据放置到SSD，最大程度地发挥SSD缓存作用。开启SSD读缓存功能后，数据节点将热点数据拷贝至SSD缓存中。相较于机械硬盘，SSD无寻道时间，可以大大减少热点数据的读取时间，提高热点数据的读性能。

SSD缓存读取流程包括读缓存命中和读缓存未命中。

2.4.1 读缓存命中

当客户端访问的数据在SSD缓存中时，读取请求将由SSD直接返回至内存，称之为读缓存命中。以POSIX协议访问为例，读取流程如图2所示。具体如下：1）客户端向索引节点请求文件的位置；2）索引节点返回文件元数据信息；3）客户端计算得到文件存储的数据节点位置，并向数据节点的内存发送读请求；4）若数据未在内存中，继续向本节点的SSD发送

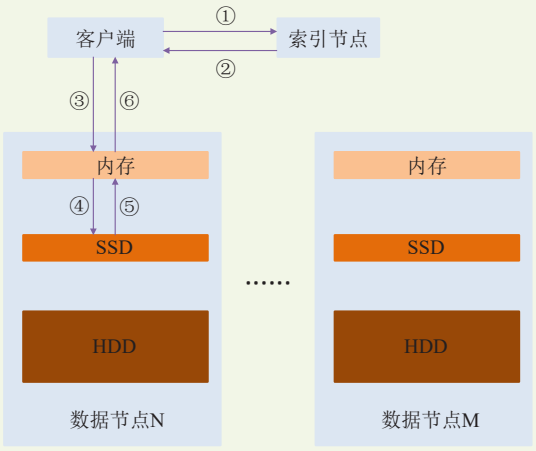

图2 读缓存命中流程

读请求；5）SSD缓存命中后，将数据从SSD读取到内存中；6）内存将读取到的数据返回到客户端。

2.4.2 读缓存未命中

当客户端访问的数据不在SSD缓存中时，读取请求需要从机械硬盘返回至内存，称之为读缓存未命中。以POSIX协议访问为例，读取流程如图3所示。具体如下：1）客户端向索引节点请求文件的位置；2）索引节点返回文件元数据信息；3）客户端计算得到文件存储的数据节点位置，并向数据节点的内存发送读请求；4）若数据未在内存中，继续向本节点的SSD发送读请求；5）SSD缓存未命中，将结果返回给内存；6）内存下发I/O请求至机械硬盘；7）机械硬盘将数据返回至内存；8）内存将数据写入到SSD，作为热点数据缓存；若SSD作为缓存的容量已满，将根据LRU算法将之前写入的部分数据抛弃；9）内存将数据返回给客户端。

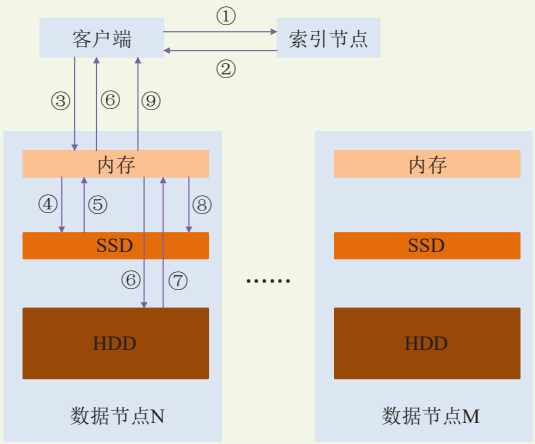

图3 读缓存未命中流程

针对逐时雷达同化预报系统的读操作，一方面将SSD盘作为二级缓存池，显著降低存储系统的响应时间，有效提高数据的访问频率；另一方面通过智能算法将访问频度高的随机小文件热点数据存放到SSD上，应用程序再次访问该数据时，可以直接从SSD上获取，提升应用程序的访存性能。

3 优化方案及测试

3.1 HAPS 算例简介

HAPS 1 km算例为单层嵌套网格，水平网格数为1281×961，垂向层数为70层，时间步长为6 s，积分时长为24 h，输入文件大小共为1.549 G，输出设置为每半小时输出模拟结果，每次输出文件大小为8.1 G，共输出文件49个，输出文件大小共为396.9 G。

3.2 测试环境

1）硬件配置

- 单节点2颗Intel Xeon Gold 6142 CPU；
- 单节点内存：192 GB；
- 通信网络：100 Gbps EDR infiniband
- 分布式并行文件存储系统

2）软件配置

- Intel compiler 2017
- Intelmpi 2017
- Netcdf 4.4.0
- Pnetcdf 1.8.1
- Slurm作业调度系统

3.3 优化方案及关键技术参数

1）MPI-IO库优化：在HAPS应用中调用支持PNETCDF库，提供集合I/O，驱动底层MPI-IO，获得持续的并行访问性能，以提高I/O效率。应用中使用PNETCDF库，版本为1.8.1，具体验证时可在编译HAPS应用时添加-DPNETCDF选项，通过设置namelist.input中的选项io_form_history参数判定输出文件使用Netcdf方式或者并行Pnetcdf方式，设置io_form_history=2为使用串行I/O方式，设置io_form_hisory=11为使用MPI-IO方式。

2）atime优化：通过mount挂载时添加-o noatime参数实现，提升读取输入数据性能。在HAPS运行过程中，多个进程读同一个文件的场景，即为多个进程并发读取同一个文件的atime的场景。对atime的修改成为数据读取的瓶颈，因此在文件系统挂载时添加-o noatime参数关闭。

3）文件锁优化：设置文件系统内部一次I/O的块大小chunksize为HAPS 1 km算例输出文件I/O块大小，针对不同的业务，文件系统可以设置不同的chunksize，确保各种大小文件的I/O性能达到最优；避免浪费网络及磁盘资源，提升读写效率。

HAPS模式调用MPI库读写数据的场景，文件系统通过提升锁服务线程个数并且在客户端开启锁缓存的方式提升加锁性能。通过专门的锁服务处理线程提升服务端锁服务处理能力，并且降低对其他请求的影响。客户端增加锁缓存，进一步降低服务端服务压力。

4）缓存命中率优化： 该文件系统利用SSD盘对随机小文件读取速度快的特点，将SSD盘组成介于HDD与内存之间的二级缓存池。在存储系统的数据节点上配置一块或多块的SSD，通过配置参数设定作为缓存资源的SSD容量。SSD只能加速所在节点上的数据访问，SSD故障后不影响数据正确性，拔掉SSD缓存盘后再插入，其上的缓存数据会自动被清除。

通过智能算法将访问频度高的随机小文件热点数据存放到SSD上，应用程序再次访问该数据时，可以直接从SSD上获取。由于SSD盘的数据读取速度远远高于机械硬盘，因此可以显著缩短热点数据的响应时间，提高缓存命中率，提高系统I/O的性能。

3.4 测试验证方案及步骤

3.4.1 验证方案

1）HAPS使用串行I/O输出方式，统计I/O时间及整个模式的墙钟时间；

2）HAPS使用串行I/O输出方式，对并行文件系统进行调优，统计I/O时间及整个模式的墙钟时间

3）HAPS使用MPI-IO输出方式，统计I/O时间及整个模式的墙钟时间；

4）HAPS使用MPI-IO输出方式，同时对并行文件系统进行调优，统计I/O时间及整个模式的墙钟时间。

通过以上四种运行方式的对比，验证提出的并行I/O优化技术的效果。

3.4.2 验证步骤

编译NETCDF库、PNETCDF库、及HAPS应用，生成haps.exe可执行文件。

编写运行验证运行脚本haps.sh：

```
#!/bin/bash
#SBATCH -J HAPS
#SBATCH --comment=HAPS
#SBATCH -p normal
#SBATCH -o log.%j
#SBATCH -e log.%j
#SBATCH -t 12:00:00
#SBATCH -N 50
#SBATCH -n 1500
#SBATCH --exclusive
module load compiler/intel/composer_xe_2017.2.174
module load mpi/intelmpi/2017.2.174
export I_MPI_FABRICS=shm:dapl
export I_MPI_DAPL_UD_PROVIDER=ofa-v2-mlx5_0-1u
export I_MPI_DAPL_PROVIDER=ofa-v2-mlx5_0-1u
export I_MPI_LARGE_SCALE_THRESHOLD=8192
export FORT_BUFFERED=1
echo $(date +%H:%M:%d)
mpirun ./haps.exe
echo $(date +%H:%M:%d)
```

通过提交sbatch haps.sh脚本进行优化验证。

3.5 效率验证

使用1500个处理器核心数运行算例，运行结果如图4所示，其中：输入时间=读取初始场文件时间+读取边界场文件时间，输出时间=wrfout文件输出时间的总和，计算时间=墙钟时间－输入时间－输出时间。

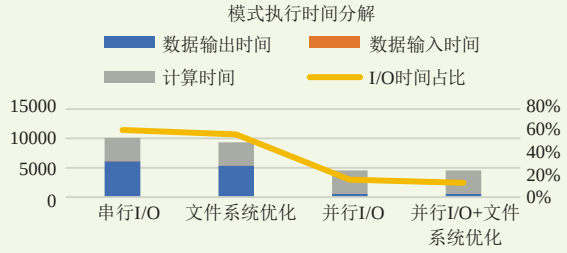

图4 墙钟时间分解

运行HAPS 1 km时使用串行I/O方式，由于输出文件数据量较大，I/O时间占总墙钟运行时间的61%。I/O成为整个模式的性能瓶颈。因此应用并行I/O技术并对并行文件系统底层软件进行优化，首先HAPS仍使用串行I/O输出方式，对分布式文件系统进行调优，I/O时间由61%降低为 57%；其次单独应用MPI-IO技术后进行测试，实现并行输出气象要素，I/O时间占比由61%降低为16%；最后测试对分布式文件系统进行调优并应用MPI-IO技术，I/O时间占比由16%降低为12.2%，由于输入文件大小仅为1.549 G，因此读入部分耗时占比较低。该试验结果表明，通过该方法可显著提高模式整体运行性能。

4 结语

本文针对深圳市气象局数值计算时普遍遇到的I/O性能瓶颈问题，开展提高气象数值模式计算性能的并行I/O优化技术研究。一方面，在HAPS应用中支持PNETCDF库，驱动底层MPI-IO，获得持续的并行访问性能；另一方面，针对业务特性，开展并行文件系统参数优化，通过调优分布式锁，提升MPI-IO优化技术在实际应用中的性能。通过实验验证表明，该优化方法可显著提升HAPS 1 km的整体I/O性能。

深入阅读

Gropp W, Lusk E, Doss N, et al, 1996. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing, 22(6): 789828.

Gropp W, Lusk E, Thakur R, 1999. Using MPI-2: Advanced features of the message passing interface. Cambridge: MIT Press.

Thakur R, Gropp W, Lusk E, 1999. On implementing MPI-IO portably and with high performance. In Proceedings of the Workshop on Input/Output in Parallel and Distributed Systems.

（作者单位：邓平，深圳市气象局；张庆花、张玉龙，曙光信息产业（北京）有限公司）