Lustre Investigation Report

FMS and NetCDF optimization

Author	Date	Description of Document Change	Client Approval By	Client Approval Date
LiuYing	May 21, 2008	Create the document		
LiuYing	May 28, 2008	Revise per the current progress		
LiuYing	Sep.10, 2008	Update the performance results		

Problem Statement

FMS(Flexible Modeling System) is a software framework for supporting the efficient development, construction, execution, and scientific interpretation of atmospheric, oceanic, and climate system models. In ORNL tests, FMS was reported to have performance problems with NetCDF on liblustre. It took 2000 processor cores about 3 hours to write 200 GB data with each processor writing 100MB.

Brief Introduction to FMS

There are four layers in the architecture of FMS - coupler layer, model layer, distributed grid layer and machine layer. The MPP(Massive Parallel Processing) modules in distributed grid layer and machine layer, form the parallel architecture of FMS. It also calls NetCDF interfaces to implement its IO.

Approach

To find the bottlenecks and improve the performance, we investigated the FMS I/O pattern, NetCDF performance tuning and the FMS performance study.

1. **FMS I/O pattern**

   a) Parallel I/O mode

   There are two kinds of parallel I/O modes in mpp_io_mod. They are described by two parameters, *threading* and *fileset.*

   - Single-threaded I/O: a single process acquires all the data and writes it to one file. This I/O mode can provide acceptable performance for small scale applications, but when the number of the processes increase, the performance may decrease because one I/O process has very bad scalability.
   - Multi-threaded, multi-fileset I/O: One file one process, which is also called file-per-process(FPP) mode. Usually this mode can achieve good performance in run time, but the end users have to do an offline post-processing to combine those independent files, and the system has to pay extra overhead to manage the files and lots of metadata.

     The FMS application uses FPP mode in our test.

   b) I/O pattern

   In FMS, fms_io_exit() is called after all fields have been written to temporary files, then the resulting NetCDF files are created. Since FMS writes data by NetCDF interfaces, its I/O pattern depends on NetCDF's access pattern. The I/O

data size written each time is determined by the NetCDF block size. If the data size is bigger than NetCDF block size, the data will be divided into smaller chunks of the block size, otherwise the small data won't be written out until a block size worth of data is collected. The NetCDF block size is discussed in the next section.

c) I/O operations

The main I/O operations in MPP modules include mpp_open(), mpp_close(), mpp_write and mpp_flush(). The data processing from mpp modules to NetCDF interfaces, then to POSIX is described in Figure 1.

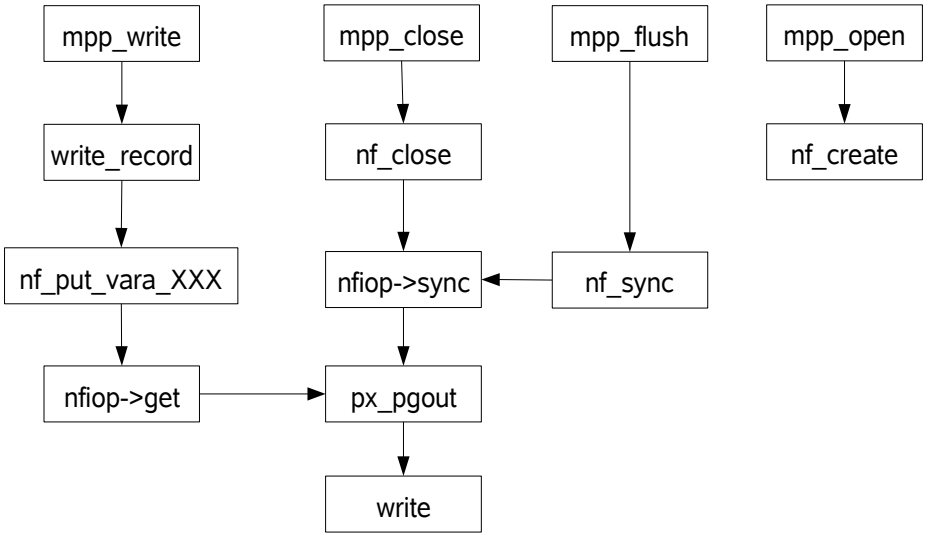

Figure 1. mpp_io operations

2. NetCDF

a) Stripe alignment

Lustre distributes files across the OSTs as per the stripe size. If I/O data is not aligned with the stripe size, extent lock conflicts will occur and cause performance issues. The NetCDF code for defining the block size for reads and writes to files is as shown below.

```
/*
 * What is the preferred I/O block size?
 */
static size_t blksize(int fd)
{
#if defined(HAVE_ST_BLKSIZE)
        struct stat sb;
        if (fstat(fd, &sb) > -1)
       {
                if(sb.st_blksize >= 8192)
                        return (size_t) sb.st_blksize;
                  return 8192;
        }
        /* else, silent in the face of error */
#endif
        return (size_t) 2 * pagesize();
}
```

The hint in NetCDF create/open operation can be used to set blksz for performance tuning.

int nc__create(const char path[], int cmode, size_t initialsize, size_t* chunksize, int* ncid)
int nc__open(const char path[], int mode, size_t* chunksize, int* ncid)

b) stripe_count

Bug(14010) in Lustre for FPP mode reports that if stripe_count is set to -1 by default, the performance might be very bad, because the file is distributed across all the OSTs, which will introduce many extent lock conflicts. If stripe_count is set to 1, which means each file is stored in only one OST, the performance could increase.

We compared these two different conditions as shown in Table 1. The recommendation is to set the stripe_count equal to 1.

Test cases and Results

We tested the FMS application on ORNL Jaguar.

1. NetCDF

We added NetCDF interfaces in the IOR benchmark to simulate FMS behavior and to check if NetCDF has some performance bottlenecks with Lustre.

a) stripe_count

As mentioned above, for FPP mode, stripe_count should be set 1. A test was performed in FPP mode on 128 clients and stripe count was set to -1 and 1 respectively. We used the following IOR command “IOR -a API -b 1m -t 1m -F -q -w -v -o testfile”. The result is as shown in Table 1.

Table 1. Write performance of NetCDF

API	stripe_count	B.W.(MB/s)	open(s)	write(s)	close(s)
NetCDF	-1	127.96	0.992010	0.894060	0.892494
	1	**4277**	0.025051	0.022407	0.018789

Obviously, the write bandwidth of stripe_count=1 is more than 30x of the one of stripe_count=-1, and the time required for open, write and close operations with stripe_count=1 is much smaller than those with stripe_count=-1.

b) Chunk size

The NetCDF blksize on Jaguar is 1M, same as the stripe size, so there is no obvious performance issue.

2. FMS

We tested FMS on Jaguar with stripe_count=1, ost_num=72, ntiles=6, npz=24, days=2, dt_atmos=1800 and test_case=13 in atmosphere model. Ntiles=6 means we use cubed-sphere FV core in FMS. The runtime results of each module and I/O operation is shown in Table 2 and and the output file size and aggregation I/O bandwidth is reported as well.

Table 2. FMS performance results

layout={npes_x,npes_y}	{8,8}	{14,14}	{2,2}	{2,2}	{2,2}
npx=npy	81	155	49	81	155
nprocs	384	1176	24	24	24
cells/proc	102	122	600	1640	6006
Total runtime(s)	9.1760	22.3424	17.2550	60.2662	566.0097
FV_RESTART(s)	0.0846	0.2681	0.1276	0.3427	1.2389
FV_DYNAMICS(s)	4.7660	7.6858	16.6060	59.0096	562.1617
COMM_TOTAL(s)	2.5270	4.5569	2.6906	5.5573	26.8538
C_SW(s)	0.3104	0.5144	1.7746	6.7041	89.2150
D_SW(s)	1.2602	2.0012	7.4801	31.0152	336.1980
TRACE_2D(s)	1.2753	2.1418	1.0829	2.1742	11.1875
COMM_TRAC(s)	0.6852	1.1815	0.4204	0.5095	1.3150
REMAPPING(s)	0.0927	0.0957	1.0599	2.8463	10.2791
OMEGA_DEL2(s)	0.0470	0.0850	0.0942	0.1504	0.4266
FV_DIAG(s)	0.3672	0.4904	0.2089	0.5250	2.0995
Open (aggregation) (s)	380.96	2241.11	0.63	0.394537	0.425904
Unlink (aggregation) (s)	351.66	6709.69	1.12	0.861732	0.830317
write(aggregation) (s)	1.646247	4.811195	0.167909	0.411026	1.016942
close(aggregation) (s)	5.640173	37.057911	0.029434	0.016724	0.020184

1 atmos_daily file (bytes)	40044	34184	327468	872364	3175688
1 grid_spec file (bytes)	17992	17880	22744	31320	66840
1 surf_hourly file (bytes)	22380	20964	91644	222844	777252
Aggregation B.W.(filesize/IOtime) (MB/s)	11.3	10.71	124.64	367.47	962.84

We can scale-up or speed-up (defined in the following) the problem to evaluate the system performance, including the capability of both I/O and computation. However, only I/O performance is discussed in this report.

Here are some definitions:

- problem size: the number of the cells, $ncells = npx \times npy \times ntiles$;
- system size: the number of the processes, $nprocs = npesx \times npesy \times ntiles$. The larger the number is, the stronger the computation capability the system has.
- workload intensity: the number of the cells per process, $wl = \frac{npx \times npy}{npesx \times npesy}$. Ideally, the larger the number is, the more computation time would be spent and the more data would be written.

a) scale-up

For the first two test cases in Table 2 - For about 100 cells per process, when the number of processes increased, the file size each process wrote was almost the same, but the total run time increased. Because the computation time couldn't cost much due to only 100 cells per process, it could be caused by the I/O operation. The results showed that $opentime/proc_{nprocs=364} = 0.99(s)$ and $opentime/proc_{nprocs=1176} = 1.91(s)$. When the number of the processes increases, the average number of the processes accessing each OST increases too and I/O access conflicts occur. Also, the aggregation I/O bandwidth shows that if many processes access the same OST, it will cause performance degradation due to competition.

a) speed-up

For the last three test cases in Table 2 - For the same system scale(nprocs=24), when problem size became larger, the workload intensity of each process grew heavier, the file size each process wrote became larger too, and the total runtime increased remarkably. But I/O operation time didn't cost much, but the computation time did. The results showed that $writetime/proc_{(npx=49)} = 0.007(s)$, $writetime/proc_{(npx=81)} = 0.017(s)$ and $writetime/proc_{(npx=155)} = 0.042(s)$. Although the total runtime for npx=npy=155 is 566 seconds, the aggregation I/O time is no more than 3 seconds. This proves that the computation cost were significant.

The analysis above shows that for a given FMS system or problem, we should consider the impact of both computation time and I/O bandwidth to find an optimal solution, as there is a trade-off between them. They determine the response time and throughput of the system together. Apparently, for the same system scale, the incrementing of the cells number will help a lot with the aggregation I/O bandwidth, but the computation time increases significantly. On the contrary, for the same problem size, the increment of the nprocs will improve the computation time, but I/O conflicts will occur.

Conclusions & Future work

In this report, the investigations on the FMS application were described. The results showed that NectCDF can achieve high performance in FPP mode with stripe_count=1, and we have not found any obvious I/O performance degradation in our large scale test on Jaguar. The experiments show the evident trade-off between computation time and I/O performance when we discuss the system scale and problem size.
Although we have spent some time in studying cube-sphere FV core in FMS, we are not sure if we can reproduce any performance problem. If the performance issue occurs again or more information can be provided, we will investigate this further in the future.

Reference

1. http://www.gfdl.noaa.gov/fms/
2. http://sivo.gsfc.nasa.gov/cubedsphere_overview.html
3. William M. Putman, “Development of the Finite-volume Dynamical Core on the Cubed-sphere”, Ph.D dissertation from college of arts and sciences, the Florida State University, summer, 2007.
4. http://www.unidata.ucar.edu/software/netcdf/